Sobre a estruturação de informação em sistemas de segurança computacional: o uso de ontologia

Luciana Andréia Fondazzi Martimiano

Sobre a estruturação de informação em sistemas de segurança computacional: o uso de ontologia

Luciana Andréia Fondazzi Martimiano

Orientador: *Prof. Dr. Edson dos Santos Moreira*

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional.

USP - São Carlos
Agosto de 2006

Ao Richard, um marido maravilhoso e um amigo incondicional. Meu grande amor.

Agradecimentos

A Deus pela minha vida.

Ao meu marido, Richard, pelo apoio e amor incondicionais, e pelos sacrifícios que ele realizou para que eu pudesse terminar este trabalho.

Aos meus pais, Darcy e Cândido, e aos meus irmãos, Fabiano e Renato, pelo amor e carinho.

Ao meu orientador, Edson, pelo apoio, pelas idéias e pela amizade.

Aos meus alunos, Matheus, Paula e Rafael, pela grande ajuda nas implementações realizadas neste trabalho.

Aos amigos Renato e Flávia, pelas ótimas discussões que tivemos sobre ontologias.

Ao meus amigos Adenilso e Elaine, pela grande ajuda com o LaTeX.

A todos os amigos que me acompanharam durante todo este tempo, principalmente nos momentos de descontração e festa. Obrigada por vocês existirem.

Ao Programa de Pós-Graduação do ICMC-USP.

Resumo

Como a quantidade e a complexidade de informações disponíveis sobre incidentes de segurança é crescente, as tarefas de manipular e gerenciar essas informações tornaram-se bastante custosas. Diversas ferramentas de gerenciamento de segurança estão disponíveis para auxiliar os administradores. Essas ferramentas podem monitorar tudo que entra e saí de uma intranet, como os *firewalls*; podem monitorar o tráfego interno da rede para saber o que está acontecendo e detectar possíveis ataques, como os sistemas de detecção de intrusão (SDIs); podem varrer arquivos em busca de códigos maliciosos, como os antivírus; podem criar filtros de *emails* para evitar *spams*, vírus ou *worms*; ou podem varrer uma rede em busca de vulnerabilidades nos sistemas, como os *scanners* e os agentes móveis inteligentes. Essas ferramentas geram uma grande quantidade de *logs* com informações que são coletadas e armazenadas em formatos próprios e diferentes. Essa falta de um formato único para armazenar as informações de incidentes de segurança, faz com que o trabalho dos administradores fique ainda mais difícil, pois eles/elas devem ser capazes de entender todos esses formatos para identificar e correlacionar informações quando, por exemplo, há um ataque ou uma invasão em andamento. Esta tese descreve o projeto e o desenvolvimento de ontologias para representar em uma estrutura padronizada informações sobre incidentes de segurança. A ontologia desenvolvida é denominada OntoSec - Security Incident Ontology. Este trabalho cobre: (i) como utilizar ontologias para compartilhar e reusar informações sobre incidentes; (ii) como correlacionar incidentes por meio de ontologias; (iii) como facilitar a interoperabilidade entre diferentes ferramentas de segurança; (iv) a modelagem de um sistema de gerenciamento de incidentes com base na ontologia; e (v) o processo de avaliação da ontologia desenvolvida. Além disso, a OntoSec pretende apoiar as decisões gerenciais realizadas pelos administradores quando problemas de segurança acontecem, possibilitando que essas decisões sejam tomadas de maneira mais eficiente e eficaz.

Abstract

As the amount and the complexity of security incidents information have grown exponentially, managing and manipulating these information have become more expensive. Several security tools can be used to assist the administrators in performing these tasks. These tools can monitor what comes from Internet and goes to it, as the firewalls do; they can monitor the intranet traffic, as usually is done by an Intrusion Detection System (IDS); they can search for malicious codes in files or emails, as made by the antivirus; they can create filters to process spams, viruses or worms; or they can scan the intranet for vulnerabilities, as the scanners and the intelligent agents. These tools collect and store a great amount of information, using different formats. This lack of unique commonly agreed formats to store information about security incidents, make the administrators' job even harder, because they have to be able to understand all these formats to identify and to correlate information when, for instance, there is an attack or an invasion in progress. In this thesis I describe the design and development of ontologies to represent in a standard structure information about security incidents. The ontology developed is named OntoSec - Security Incident Ontology. This work covers: (i) how to use ontologies to share and reuse information about incidents; (ii) how to make it easier to correlate incidents; (iii) how to make it possible the interoperability amongs security tools; (iv) modeling of a security incident management system based on OntoSec; and (v) evaluation process of the ontology that has been developed. Besides that, the OntoSec aims to support the decisions made by the administrators when security problems happen, making the process more efficient and effective.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ABNT: Associação Brasileira de Normas Técnicas
ABox: *Assertional Box*
API: *Application Programming Interface*
ARPANET: *Advanced Research Projects Agency Network*
ARP: *Address Resolution Protocol*
BNF: *Backus-Naur Form*
BS: *British National Standard*
CAIS: Centro de Atendimento a Incidentes de Segurança
CCE: Centro de Computação Eletrônica
CERT/CC: *Computer Emergency Response Team/Coordination Center*
CERT.br: Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança
CNA: *Candidate Numbering Authority*
CSIRT: *Computer Security Incident Response Team*
CPU: *Central Processing Unit*
CUPS: *Common UNIX Printing System*
CVE: *Common Vulnerabilities and Exposures*
DAML: *DARPA Agent Markup Language*
DARPA: *Defense Advanced Research Projects Agency*
DDoS: *Distributed Denial of Service*
DHCP: *Dynamic Host Configuration Protocol*
DL: *Description Logics*
DNS: *Domain Name Service*
DoS: *Denial of Service*
DTD: *Document Type Definition*
E/S: *Entrada/Saída*
FTP: *File Transfer Protocol*
HTML: *Hypertext Markup Language*
HTTP: *Hypertext Transfer Protocol*
HTTPS: *Secure Hypertext Transfer Protocol*
ICMP: *Internet Control Message Protocol*
IDMEF: *Intrusion Detection Message Exchange Format*
IDS: *Intrusion Detection System*
IDWG: *Intrusion Detection Working Group*
IEC: *International Electrotechnical Commission*
IEEE: *Institute of Electrical and Electronics Engineers*
IETF: *Internet Engineering Task Force*

IMAP: *Internet Message Access Protocol*
IMC: *Internet Mail Consortium*
I/O: *Input/Output*
ISO: *International Standards Organization*
LAN: *Local Area Network*
LDAP: *Lightweight Directory Access Protocol*
MAC: *Medium Access Control*
MIT: *Massachusetts Institute of Technology*
NBR: Norma Brasileira
NetBios: *Network Basic Input/Output System*
NIS: *Network Information Service*
NVD: *National Vulnerability Database*
OML: *Ontology Markup Language*
OIL: *Ontology Inference Layer*
OKBC: *Open Knowledge Base Connectivity*
OntoSec: Ontologia de Incidentes de Segurança (*Security Incident Ontology*)
OntoSec: *Ontology for Security*
OSI: *Open Systems Interconnection*
OWL: *Web Ontology Language*
POP: *Post Office Protocol*
RAM: *Random Access Memory*
RDF: *Resource Description Framework*
RDFS: *RDF Schema*
RDQL: *RDF Data Query Language*
RFC: *Request For Comment*
RNP: Rede Nacional de Ensino e Pesquisa
RPC: *Remote Procedure Call*
RST: *Reset*
Safebots: *Software Robots*
SBC: Sistemas Baseados em Conhecimento
SEI: *Software Engineering Institute*
SHOE: *Simple HTML Ontology Extensions*
SMTP: *Simple Mail Transfer Protocol*
SNMP: *Simple Network Management Protocol*
SOAP: *Simple Object Access Protocol*
SPARQL: *SPARQL Query Language for RDF*
SSH: *Secure Shell*
SFTP: *Secure FTP*
SSL: *Secure Socket Layer*
SQL: *Structured Query Language*
TCP/IP: *Transmission Control Protocol/Internet Protocol*
TBox: *Terminological Box*
Telnet: *Telephone Network*
TIC: Tecnologia da Informação e Comunicação

TLS: *Transport Layer Security*
TTL: *Time to Live*
UDP: *User Datagram Protocol*
UML: *Unified Modeling Language*
URI: *Universal Resource Identifier*
URL: *Universal Resource Locator*
USP: Universidade de São Paulo
XML: *Extensible Markup Language*
XMLS: *XML Schema*
XOL: *Ontology eXchange Language*
W3C: *World Wide Web Consortium*
WINS: *Windows Internet Naming Service*
Web: *World Wide Web*
WWW: *World Wide Web*

Capítulo

1

Introdução

1.1 Considerações Iniciais

A informação tornou-se algo imprescindível para o sucesso de qualquer organização, seja militar, empresarial ou acadêmica. Com a crescente e constante evolução de diferentes tecnologias, tais como: negócio e comércio eletrônicos, telecomunicações e computação, e a própria Internet, a quantidade de informação disponível tem aumentado de maneira exponencial. Bancos de dados permitem que muitos dados sejam armazenadas e possam ser acessados para diferentes fins, gerando uma grande quantidade de informação. No entanto, algumas questões que surgem são: Como gerenciar toda essa informação? O que é e o que não é importante? Como proteger essas informações?

Por outro lado, a preocupação com a segurança computacional, principalmente com a segurança da informação propriamente dita, tem se tornado mais presente nas organizações. Como a quantidade de dados e informações disponíveis de segurança também é crescente, as tarefas de manipular e gerenciar esses dados e informações tornaram-se bastante custosas. Tecnologias como Bancos de Dados, *Data Warehousing*, Mineração de Dados (*Data Mining*) e Ferramentas de Gerenciamento de Segurança têm auxiliado na execução dessas tarefas, com o intuito de facilitar a geração de conhecimento que possa ser utilizado para que administradores de sistemas tomem decisões de maneira mais eficiente.

Diversas ferramentas de gerenciamento de segurança que auxiliam os administradores estão disponíveis. Essas ferramentas podem monitorar tudo que entra e sai de uma intranet, como é o caso dos *firewalls*; podem ainda monitorar o tráfego interno da rede

para saber o que está acontecendo e detectar possíveis ataques, como é o caso dos sistemas de detecção de intrusão (SDIs); podem varrer arquivos em busca de código malicioso, como é o caso dos antivírus; podem criar filtros de *emails* evitando *spams*, vírus ou *worms*, por exemplo; ou podem varrer uma rede em busca de vulnerabilidades nos sistemas, como os *scanners* e os agentes móveis inteligentes. Essas ferramentas geram uma grande quantidade de *logs* com informações que são coletadas e armazenadas em formatos próprios.

Essa falta de um formato único em armazenar informações de segurança, faz com que o trabalho do administrador de segurança fique mais difícil, pois ele deve ser capaz de entender todos esses formatos para identificar, por exemplo, quando há um ataque ou uma invasão em andamento. Além disso, o administrador deve se utilizar de seu conhecimento tácito para identificar que um determinado incidente de segurança está relacionado com um incidente prévio, ou seja, estabelecer uma correlação entre os incidentes de segurança depende do conhecimento do administrador[1].

1.2 Contexto e Motivação

Em sistemas de segurança computacional, o administrador tem em mãos uma grande quantidade de dados e informações que devem ser gerenciados e analisados a fim de tomar uma decisão caso o sistema seja invadido ou atacado. Esses dados podem ser de fontes diretamente relacionadas à segurança, tais como: *logs* de acesso às máquinas que compõem o sistema, *logs* do *firewall* e alertas de vulnerabilidades[2], e/ou de fontes que podem fornecer dados que auxiliam na elaboração de informações que podem caracterizar uma quebra de segurança, tais como: estatísticas de taxas de utilização dos processadores que compõem o ambiente do sistema, estatísticas da utilização da largura de banda, sobrecarga de memória, etc. Assim, torna-se, muitas vezes, impraticável o processamento da informação em tempo hábil para a tomada de decisão antes que um incidente alcance grandes proporções.

Diferentes institutos de pesquisas têm realizado esforços no sentido de catalogar e classificar dados e informações relacionados à segurança. O projeto CVE (*Common Vulnerabilities and Exposures*), desenvolvido no Instituto Mitre[3], apresenta um formato

[1] Neste trabalho, um incidente de segurança é um **ato de violar uma política de segurança**, definição do CERT/CC (*Computer Emergency Response Team/Coordination Center*). `http://www.cert.org/`.

[2] Vulnerabilidade é qualquer fraqueza que pode ser explorada para se violar um sistema ou suas informações (ISO, 1989).

[3] `http://cve.mitre.org/`.

único para armazenar vulnerabilidades que facilita a identificação de uma mesma vulnerabilidade em diferentes ferramentas (Mann e Christey, 1999). Além do CVE, a base de dados do NVD (*National Vulnerability Database*) oferece uma interface de busca às vulnerabilidades presentes no CVE, com informações a respeito de correções (*patches*), softwares e referências a respeito das vulnerabilidades. O CERT, criado como parte do Instituto de Engenharia de Software (SEI - *Software Engineering Institute*) da Universidade de Carnegie Mellon nos EUA (Estados Unidos da América), tem como intuito principal prover uma organização capaz de coordenar respostas a problemas de segurança na Internet, armazenando informações relacionadas a vulnerabilidades, incidentes de segurança, alertas de vírus, entre outras.

No Brasil, o CAIS (Centro de Atendimento a Incidentes de Segurança)[4], centro ligado à Rede Nacional de Ensino e Pesquisa (RNP), e o CERT.br (Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil)[5], atualmente mantido pelo Comitê Gestor da Internet no Brasil, são os principais responsáveis por armazenar informações de segurança no *backbone* brasileiro. Esses institutos são fontes de informações importantes que podem auxiliar o administrador de segurança.

Apesar desses institutos terem bases de dados bastante ricas e importantes, as informações registradas e armazenadas não são totalmente públicas. Somente os *sites*, organizações ou produtos relacionados diretamente aos incidentes de segurança têm acesso completo às informações. Essa política é adotada para proteger os envolvidos nos incidentes. Além disso, nenhuma dessas bases divulga as possíveis correlações que possam existir entre problemas com segurança, como vulnerabilidades e incidentes, por exemplo. Esse último aspecto dificulta a geração de conhecimento a respeito de incidentes de segurança, e dificulta o trabalho dos administradores de segurança.

Para facilitar essa correlação, ontologias podem ser utilizadas no sentido de estabelecer um formato para representar e armazenar informações a respeito de um domínio de conhecimento. Uma ontologia, segundo Gruber (1995), é **"especificação formal e explícita de uma abstração, uma visão simplificada de um domínio de conhecimento"**. A ontologia provê uma compreensão comum sobre um domínio, visando o reúso e o compartilhamento de conhecimento.

Donner (2003), no editorial da Revista IEEE (*Institute of Electrical and Electronics Engineers*) *Security & Privacy*, ressalta a importância de se desenvolver uma ontologia para o domínio de segurança, descrevendo os principais conceitos existentes e seus

[4] `http://www.rnp.br/cais`.

[5] Anteriormente o CERT.br era denominado *NBSO (NIC BR Security Office)/Brazilian CERT*. `http://www.cert.br`.

relacionamentos. Brandão (2004) investigou o uso de ontologias para classificar alertas de vulnerabilidades gerados por diferentes ferramentas de segurança. Undercoffer *et al.* (Undercoffer et al., 2003) desenvolveram uma ontologia para representar os alertas de ataques gerados por sistemas de detecção de intrusão. Raskin et al. (2001) estudaram como o uso de ontologia pode formalizar a conceituação e a terminologia de incidentes de segurança. Schumacher (Schumacher, 2003) desenvolveu um modelo conceitual de políticas de segurança utilizadas no dia a dia do gerenciamento de incidentes de segurança. Os trabalhos de Raskin *et al.* e Schumacher propõem uma visão abstrata e de alto nível do gerenciamento de incidentes de segurança.

Este trabalho propõe o desenvolvimento e o uso de uma ontologia para representar de maneira padronizada as informações sobre incidentes, facilitando (i) o compartilhamento e o reúso dessas informações, (ii) o gerenciamento de incidentes de segurança, (iii) a geração de conhecimento sobre incidentes, (iv) a correlação entre os incidentes, (v) a interoperabilidade entre ferramentas de segurança, e (vi) o processo de validação e avaliação de ontologias. A ontologia proposta e desenvolvida neste trabalho é denominada OntoSec - Ontologia de Incidentes de Segurança. Diferentemente dos trabalhos desenvolvidos por Schumacher e Raskin *et al.*, a OntoSec define de maneira mais concreta como as informações de segurança podem ser representadas no nível gerencial de uma organização.

1.3 Objetivos

Os principais objetivos deste trabalho são:

- Estabelecer uma estrutura padronizada para representar informações sobre incidentes de segurança.
- Realizar o mapeamento automático das informações sobre incidentes de segurança em uma linguagem padrão de representação, facilitando a interoperabilidade entre diferentes ferramentas de segurança.
- Desenvolver uma aplicação de consulta que permita extrair informações da ontologia, mostrando que é possível realizar uma correlação entre incidentes de segurança.
- Demonstrar o uso de ontologias como facilitador de compartilhamento e reúso de informações, e de conhecimento sobre incidentes de segurança.
- Modelar uma ferramenta de gerenciamento de incidentes de segurança a partir da ontologia de incidentes de segurança.

1.4 Organização do Trabalho

Este trabalho está organizado da seguinte maneira:

O Capítulo 1 apresenta o contexto, a motivação e os principais objetivos deste trabalho. Os Capítulos 2 e 3 apresentam, respectivamente, os principais conceitos e ferramentas de segurança da informação e os principais conceitos sobre ontologias. O Capítulo 4 descreve o processo de desenvolvimento da OntoSec, enquanto que o Capítulo 5 descreve o processo de avaliação da ontologia. O Capítulo 6 apresenta como a OntoSec pode auxiliar no gerenciamento de incidentes de segurança. O Capítulo 7 apresenta as considerações finais, as contribuições deste trabalho e as propostas de trabalhos futuros. Por fim, os Apêndices A e B apresentam, respectivamente, o vocabulário de conceitos e de relacionamentos da OntoSec e os boletins de segurança utilizados como exemplos para instanciar a ontologia.

Capítulo

2

Segurança Computacional

2.1 Considerações Iniciais

Ao longo da evolução da Internet, iniciada no final dos anos 60 com o surgimento da ARPANET (*Advanced Research Projects Agency Network*), uma das maiores preocupações tem sido com a segurança das informações que trafegam nessa rede mundial. A crescente preocupação por segurança nas organizações está diretamente relacionada ao fato de que, atualmente, o bem mais precioso dessas organizações é a informação. Uma vez que o conhecimento é gerado a partir dessa informação, tornou-se crucial para as organizações estabelecer meios que auxiliem na proteção dos sistemas computacionais contra ações não autorizadas.

Neste capítulo são discutidos os principais conceitos e ferramentas na área de segurança computacional, além de apresentar alguns esforços da comunidade científica em desenvolver maneiras mais eficientes para auxiliar os profissionais da área na tarefa de gerenciar a segurança da informação[1].

2.2 Conceitos e Definições na área de Segurança

Segundo Gollmann (1999), segurança computacional trata a prevenção e a detecção de ações não autorizadas a um sistema computacional. Garfinkel e Spafford (1996) definem

[1]Não são discutidos fatores relacionados à segurança física (ou do ambiente).

segurança computacional da seguinte forma: "um sistema é seguro se for possível confiar que esse sistema se comporta como o esperado". Ambas as definições se diferem no modo de medir segurança. Para Gollmann, a segurança é medida pela falta de ações de intrusões ou de ataques bem sucedidos, enquanto que para Garfinkel e Spafford a segurança está relacionada com a capacidade do sistema em garantir continuidade de serviços após uma intrusão ou ataque.

De qualquer forma, a necessidade de proteção deve ser definida a partir das possíveis ameaças e riscos que uma organização pode sofrer. Cabe à organização, portanto, definir o que pode e o que não pode ser permitido em termos de segurança. Assim, uma política de segurança deve ser estabelecida levando em consideração o grau de segurança que se deseja alcançar. Além disso, uma política de segurança deve determinar como cada parte do sistema deve funcionar e deve ser utilizada, e quais os deveres e direitos de cada elemento que utiliza esse sistema, e como e quais são os ativos que devem ser protegidos. Uma vez definidos quais e como os ativos devem ser protegidos, a política de segurança pode especificar qual o grau de confidencialidade, integridade, disponibilidade, controle e autenticação esperado. A partir da política de segurança, é possível ainda estabelecer quais ações devem ser executadas caso problemas de segurança ocorram e quais ferramentas podem ser utilizadas para garantir o grau de segurança definido. Assim, o nível de confiança do sistema é obtido pela comparação entre a forma de como ele se comporta e a forma especificada de como ele deve se comportar.

Confidencialidade requer que as informações sejam acessadas somente por pessoas autorizadas a fazê-lo. **Integridade** requer que as informações não sejam alteradas acidental ou maliciosamente. **Disponibilidade** requer que o sistema computacional funcione sem degradação e forneça os recursos aos usuários autorizados quando necessário (Kumar, 1995).

Autenticidade é utilizada para se comprovar que o usuário é realmente quem diz ser. Um dos recursos mais utilizados e conhecidos por conferir autenticidade é a senha, a princípio, conhecida somente pelo usuário. Atualmente, recursos biométricos[2] estão sendo utilizados para garantir maior segurança nos sistemas computacionais. **Autorização** é o ato de determinar se um usuário em particular (ou um sistema computacional) tem o direito de executar determinada tarefa, tal como leitura de um arquivo ou execução de um programa. Geralmente, usuários estão divididos em grupos diferentes com direitos diferentes. Autenticidade e autorização são sempre empregadas em conjunto: um usuário precisa ser autenticado antes de poder executar tarefas que ele esteja autorizado a execu-

[2]Recursos biométricos utilizam características do ser humano para identificação, tais como: impressão digital, íris ou voz.

tar. **Irretratabilidade** está relacionada com o fato de que um usuário não pode negar que executou uma determinada tarefa. Por exemplo, o resultado de uma autenticação não pode ser negado posteriormente.

Confidencialidade, integridade e disponibilidade estão relacionadas diretamente às informações do sistema computacional, enquanto que autenticidade, autorização e irretratabilidade estão relacionados diretamente às pessoas que utilizam o sistema.

2.3 Ferramentas para Gerenciamento de Segurança

Devido à facilidade com que ferramentas automáticas de invasão podem ser encontradas, invadir um sistema computacional requer cada vez menos habilidades dos intrusos. Enquanto os intrusos mais experientes estão se tornando mais eficientes, o conhecimento requerido por parte de intrusos novatos para copiar e iniciar ataques está diminuindo (Allen et al., 2000). Com isso, cada vez mais sistemas computacionais são invadidos a cada dia.

Intrusos podem ser classificados em ativos e passivos. Um intruso passivo é aquele que simplesmente monitora a rede (sistema) em busca de informações como senhas, números de cartões de créditos ou outras informações confidenciais, enquanto que um intruso ativo é aquele que atua modificando indevidamente as informações acessadas.

Segundo pesquisa realizada pela *Módulo Security* (Módulo, 2003) com empresas brasileiras, em sua maioria Instituições Financeiras (21%) e Agências Governamentais (19%), cresce a cada ano a preocupação das empresas com a proteção de suas informações, aumentando também a adoção de controles para minimizar os riscos resultantes de ameaças e vulnerabilidades. A cada ano, a segurança computacional tem sido tratada pelos altos executivos com mais importância e com respectivo aumento de investimentos focados na efetiva redução dos riscos operacionais. A segurança da informação está se tornando diferencial estratégico para as organizações.

A pesquisa da Módulo também revela um fato interessante: a principal ameaça à informação são os funcionários insatisfeitos, com 64%. Por outro lado, 48% das empresas apontaram os *crackers* como os principais responsáveis por invasões e ataques. A pesquisa ainda revela que 42% das empresas tiveram problemas com a segurança da informação nos últimos dois anos, sendo que 33% sofreram algum tipo de ataque ou invasão nos últimos 12 meses. A Internet foi apontada por 55% das empresas como o principal ponto de invasão. Segundo a pesquisa, as principais ameaças são:

- Problemas com vírus.

- Funcionários insatisfeitos.
- Divulgação de senhas.
- Acessos indevidos.
- Vazamento de informação.
- Fraudes em e-mails (*Scam Phishing*).

Além dessas ameaças, ainda podem ser citadas: negação de serviço (*Denial of Service - DoS*), que caracteriza a degradação ou indisponibilidade de serviço, e os e-mails indesejados, os chamados *spams*.

2.3.1 Políticas de Segurança

Uma política de segurança é definida como um conjunto de normas, regras e práticas que regulam como uma organização gerencia e protege suas informações. As regras que definem uma política de segurança são funções das designações de sensibilidade, associadas aos recursos e informações (por exemplo: não classificado, confidencial, secreto e ultra-secreto), do grau de autorização das entidades (indivíduos ou processos agindo sob o comando de indivíduos) e das formas de acesso aceitas por um sistema. A implantação de uma política de segurança baseia-se na aplicação de regras que limitam o acesso de uma entidade às informações e recursos, com base na comparação do seu nível de autorização relativo a essa informação ou recurso, na designação da sensibilidade da informação ou recurso e na forma de acesso empregada.

A ABNT (Associação Brasileira de Normas Técnicas) definiu a norma NBR ISO/IEC 17799 (ABNT, 2001)[3] com o intuito de fornecer recomendações para a gestão da segurança da informação. Essa norma fornece recomendações para aqueles que são responsáveis pela introdução, implementação ou manutenção da segurança em suas organizações. Tem como propósito prover uma base comum para o desenvolvimento de normas de segurança organizacional e das práticas efetivas de gestão da segurança, e prover confiança nos relacionamentos entre as organizações.

Segundo a norma NBR ISO/IEC 17799, uma política de segurança provê formas de preservação da confidencialidade, integridade e disponibilidade das informações, definindo o que pode e o que não pode ser permitido em termos de segurança. Uma política de segurança deve conter, pelo menos, as seguintes orientações (ABNT, 2001):

[3]A NBR ISO/IEC 17799 é uma tradução para o português da norma britânica *British National Standard 7799* (BS 7799).

- Definição de segurança da informação, resumo das metas e escopo, e a importância da segurança como um mecanismo que habilita o compartilhamento da informação.
- Declaração do comprometimento da alta direção, apoiando as metas e princípios da segurança da informação.
- Breve explanação das políticas, princípios, padrões e requisitos de conformidade de importância específica para a organização, como por exemplo:
    1. Conformidade com a legislação e cláusulas contratuais.
    2. Prevenção e detecção de vírus e outros códigos maliciosos.
    3. Conseqüências das violações na política de segurança da informação.
    4. Definição das responsabilidades gerais e específicas na gestão da segurança da informação, incluindo o registro dos incidentes de segurança.

A política ainda deve ter um gestor que seja responsável por sua manutenção e análise crítica, de acordo com um processo bem definido. Convém que esse processo garanta que a análise crítica ocorra como decorrência de qualquer mudança que venha a afetar a avaliação de risco original, tais como um incidente de segurança, novas vulnerabilidades ou mudanças organizacionais ou na infra-estrutura técnica.

Existem basicamente duas filosofias por trás de qualquer política de segurança:

- **Proibitiva**: Tudo que não é expressamente permitido é proibido.
- **Permissiva**: Tudo que não é expressamente proibido é permitido.

Geralmente as instituições mais sensíveis à segurança, e que tenham um escopo de utilização de recursos de Tecnologia da Informação e Comunicação (TIC) bem definido, adotam a primeira abordagem. Uma política, neste caso, deve descrever exatamente quais operações são permitidas em um sistema. Qualquer operação que não esteja descrita de forma detalhada na política de segurança deve ser considerada ilegal ao sistema.

Uma quebra de segurança pode ser definida como uma ação, ou conjunto de ações, que está contra a política de segurança vigente. Essa política de segurança é criada de acordo com as necessidades particulares de cada organização. Dessa forma, o que é considerado uma quebra de segurança pode variar conforme a organização. A política de segurança define o que pode ou não pode ser feito, por quem e sob quais circunstâncias. Ela define ainda os procedimentos a serem tomados frente a uma invasão (Plano de Emergência), quais são os recursos que devem ser prioritariamente protegidos, que nível

de risco é aceitável, além de resolver questões éticas e polêmicas, como por exemplo, se o administrador ou chefe tem o direito de ler os *e-mails* dos funcionários. Uma vez que se tenha tudo definido e que os usuários e administradores estejam cientes dessa política, tem-se um quadro no qual é fácil determinar se uma ação é considerada uma quebra de segurança e que medidas podem ser tomadas (Bernardes, 2002).

A base de uma política de segurança é a definição do comportamento autorizado para os indivíduos que interagem com um sistema (Soares et al., 1995). Essa política deve incluir regras detalhadas definindo como as informações e recursos da organização devem ser manipulados ao longo de seu ciclo de vida, ou seja, desde o momento que passam a existir no contexto da organização até quando são excluídos. Muitos usuários respeitam o conjunto de regras sociais. Essas regras encorajam o respeito tanto à privacidade quanto ao ambiente de trabalho. Assim, torna-se relativamente fácil invadir um sistema no qual os usuários confiam uns nos outros. Diante disso, tornam-se necessárias a documentação e divulgação das regras que primam pela manutenção da privacidade e integridade.

2.3.2 Sistemas de Detecção de Intrusão

Detecção é definida como o ato de revelar, descobrir, captar ou perceber informação (Houaiss, 2003), enquanto que **intrusão** é qualquer conjunto de ações que comprometa a integridade, confidencialidade ou a disponibilidade do sistema, identificando agentes que estejam utilizando um sistema computacional sem autorização, e identificando pessoas que têm acesso legítimo ao sistema, mas estão abusando de seus privilégios (Balasubramaniyan et al., 1998).

Um sistema de detecção de intrusão (IDS - *Intrusion Detection System*) é caracterizado como um sistema capaz de realizar detecção de ataques conhecidos em sistemas computacionais e informar aos administradores. Como exemplo típico, tem-se o *buffer overflow*, que é caracterizado pela ação do invasor em sobrescrever dados em um *buffer* não testado de um programa com seus próprios dados maliciosos. Dependendo de quais dados são sobrescritos, o programa pode parar de funcionar ou pode mudar sua execução fazendo o que o invasor deseja.

As funcionalidades de um IDS tornam-se importantes na medida em que fornecem meios de inferir sobre o conteúdo das conexões permitidas e detectar as que apresentem um comportamento suspeito ou não condizente com a política de segurança (Ambrósio, 2002).

Segundo Balasubramaniyan et al. (1998), as principais características de um IDS são:

- Permanecer em execução continuamente com o mínimo de supervisão humana.

- Ser tolerante a falhas, ou seja, ser capaz de se recuperar caso o sistema pare de funcionar (acidentalmente ou por ações humanas).
- Resistir à subversão, ou seja, ser capaz de identificar se foi modificado por algum invasor.
- Gerar um *overhead* mínimo no sistema computacional no qual está rodando, evitando sobrecarga.
- Permitir configuração de acordo com a política de segurança.
- Ser capaz de se adaptar às mudanças no sistema e no comportamento dos usuários.
- Ser escalável para dar suporte à carga de monitorar várias estações e ainda produzir resultados confiáveis em tempo hábil para a tomada de decisões.

Existem diferentes classificações dos IDS. As principais classificações são feitas em termos de como o sistema aborda o problema de detectar intrusão e em termos do tratamento dos dados, sendo a última a classificação mais comum. Considerando a primeira classificação, existem três tipos:

- **Sistemas baseados em Rede**: Esses sistemas examinam os dados que trafegam pela rede por meio de monitoração *on-line* dos pacotes, procurando por indícios de um ataque que possa estar acontecendo. Essa monitoração é realizada por meio de sensores espalhados pela rede que capturam todos os pacotes endereçados ao segmento de rede nos quais se encontram. Geralmente, esses sensores são dispositivos passivos que têm pouco impacto no desempenho da rede. Por outro lado, esses sistemas têm dificuldade em processar pacotes em uma rede grande de tráfego intenso, ou isolada por *switch*[4], e não são capazes de analisar pacotes criptografados.
- **Sistemas baseados em Nós da Rede**: Esses sistemas são um tipo híbrido de IDS que conseguem superar algumas das limitações dos IDS baseados em rede. Assim como os IDS baseados em rede, esses sistemas também capturam pacotes à procura de indícios de ataques. No entanto, os sensores, ou micro-sensores, só se preocupam com os pacotes endereçados ao nó de rede no qual se encontram. Uma vez que esses sensores não capturam todos os pacotes da rede, eles podem ser mais rápidos e consumir menos recursos, causando um baixo *overhead* no sistema. Além disso, são adequados para segmentos de tráfego intenso ou isolados por *switch*.

[4]Dispositivo de rede que direciona tráfego

- **Sistemas baseados em *Host***: Esses sistemas utilizam informações coletadas em um sistema individual, permitindo uma análise das atividades com mais confiabilidade e precisão. Essa análise determina quais processos e usuários estão envolvidos em uma invasão específica. Ao contrário dos IDS baseados em rede, esses sistemas podem saber o resultado de uma tentativa de invasão, uma vez que eles têm acesso direto e monitoram os arquivos e processos do sistema utilizados como alvos pelos invasores. Geralmente, esses sistemas utilizam informações de duas fontes: auditoria do sistema operacional (mais protegido e com informações mais detalhadas) e *logs* do sistema (menores e mais claros, sendo mais fáceis de analisar). Esses sistemas são mais difíceis de gerenciar, uma vez que as informações precisam ser configuradas e gerenciadas para cada estação monitorada. Além disso, eles não podem detectar invasões direcionadas a toda a rede. Sistemas baseados em Aplicações são um subgrupo dos Sistemas baseados em *Host*, que utilizam os *logs* das aplicações para analisar os eventos ocorridos durante a execução dessas aplicações.

Independentemente do tipo de IDS, sua funcionalidade pode ser logicamente distribuída em três componentes: sensores, analisadores e interface. Os sensores são os responsáveis por coletar os dados para análise. As possíveis entradas para um sensor podem ser pacotes de redes ou arquivos de *logs*. Os sensores recolhem e transferem esses dados para os analisadores. Os analisadores, por sua vez, recebem dados dos sensores (ou de outros analisadores) e determinam se uma intrusão ocorreu. A saída de um analisador é a indicação de uma intrusão e, ainda, evidências que suportem essa indicação. Analisadores ainda podem fornecer uma lista de ações que podem ser tomadas quando houver uma intrusão. Já a interface permite que um usuário analise a saída do sistema ou controle o seu comportamento.

A pesquisa relacionada à detecção de intrusão evoluiu e produziu várias ferramentas nos anos 90. As pesquisas mais atuais procuram identificar alternativas para reconhecimento de padrões de invasões na tentativa de dar suporte aos IDS, tais como: redes neurais, mineração de dados e algoritmos genéticos. Outras pesquisas tentam sanar o problema da falta de um padrão dos alertas gerados pelos IDSs. O *Intrusion Detection Working Group* (IDWG) do *Internet Engineering Task Force* (IETF) desenvolveu o padrão IDMEF (*Intrusion Detection Message Exchange Format*) para troca de mensagens entre IDSs (Curry et al., 2005). Undercoffer *et al.* propõem o uso de ontologias para especificar um modelo único para representar tentativas de ataques que são identificadas por IDSs (Undercoffer et al., 2003; Undercoffer e Pinkston, 2002; Undercoffer et al., 2004). A Seção

2.4.3 descreve o padrão IDMEF e a Seção 3.4.3 descreve a ontologia desenvolvida por Undercoffer *et al.*.

Um dos IDS mais utilizados e conhecidos na comunidade de segurança é o **Snort**[5]. E como o **Snort** foi utilizado para validar a OntoSec, algumas das suas características são apresentadas a seguir.

A Ferramenta Snort

O **Snort** é um sistema de detecção de intrusão baseado em regras utilizado para observar todo o tráfego de um segmento de rede (modo promíscuo). O **Snort** tornou-se um padrão de *facto*, sendo uma ferramenta de código aberto largamente utilizada. Algumas características do **Snort** são:

- Realiza registros em *logs* (texto puro), que são os alertas.
- Permite armazenar os alertas em bancos de dados remotos ou locais, tais como MySQL[6] e PostgreSQL[7].
- É capaz de "resetar" (RST) uma conexão suspeita (ação conhecida como **buraco negro**).
- Possui um banco de regras com as principais vulnerabilidades conhecidas. Essas regras são armazenadas em arquivo ".rules", sendo um arquivo para cada tipo de protocolo (como FTP *File Tranfer Protocol*), ou serviço (como DNS *Domain Name Service*), entre outros. Essas regras são atualizadas diariamente pela comunidade de segurança.
- Facilita a personalização e a criação de novas regras.

O **Snort** gera alertas de segurança baseados em assinaturas de ataques. Os alertas são gerados quando determinados padrões são encontrados nos pacotes que trafegam pela rede. Assim, a ferramenta funciona muito bem para ataques já conhecidos. No entanto, novos ataques não podem ser detectados até que novas regras sejam criadas. Portanto, atualizações devem ser constantes. Além disso, muitos alertas podem ser falso-positivos, ou seja, a ferramenta registra um ataque que não está acontecendo. Nesse caso, cabe ao administrador filtrar o que é e o que não é um ataque real. Para evitar os falso-positivos, é preciso também criar regras mais precisas.

[5] `http://www.snort.org.`
[6] `http://www.mysql.com/.`
[7] `http://www.postgresql.org/.`

As regras do **Snort** são divididas em duas partes lógicas, "Cabeçalho" e "Opções", como mostra a Figura 2.1. No "Cabeçalho", o campo **"Ação"** define o que deve ser feito quando um pacote for capturado. As ações disponíveis são:

- *alert*: Gera um alerta e então registra o pacote.
- *log*: Apenas registra o pacote.
- *pass*: Ignora o pacote (útil para falso-positivos).
- *activate*: Alerta e executa uma regra dinâmica.
- *dynamic*: Espera um sinal de uma regra *"activate"* para ser executada.

O campo **"Protocolo"** identifica qual protocolo foi utilizado, sendo quatro os mais freqüentes: TCP (*Transmission Control Protocol*), UDP (*User Datagram Protocol*), IP (*Internet Protocol*) e ICMP (*Internet Control Message Protocol*). Os campos **"Endereço IP Origem"** e **"Porta Origem"** identificam, respectivamente, de quais IPs e portas o ataque partiu, enquanto os campos **"Endereço IP Destino"** e **"Porta Destino"** indicam, respectivamente, quais os IPs e portas alvos do ataque. Caso esses campos não tenham valores específicos e sim *"any"*, qualquer endereço IP e porta são considerados.

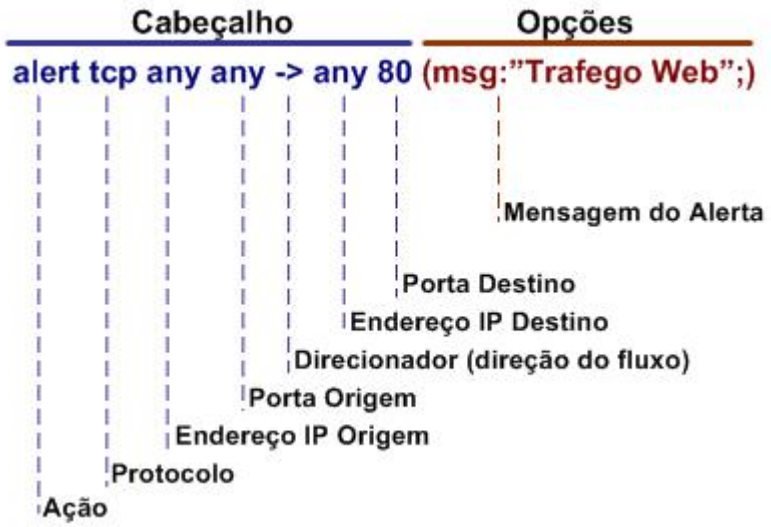

Figura 2.1: Exemplo dos campos das regras do **Snort**.

No "Opções", o campo **"Mensagem do Alerta"** especifica a mensagem que será exibida no alerta quando o pacote for capturado, e equivale a uma descrição do ataque. Além da mensagem, outras opções podem ser utilizadas: **"ID"**, que verifica o campo ID do

cabeçalho IP do pacote e é útil para detectar ferramentas *crackers* que utilizam códigos com significados especiais para troca de dados, **"TTL"** (*Time to Live*), que identifica o tempo de vida do pacote e é interessante para identificar problemas de tráfego ou tentativas de *traceroute*[8], **"Logto"**, que grava a saída de todos os pacotes capturados pela regra para um arquivo especificado, e **"Dsize"**, usada para testar o tamanho do *payload* (cabeçalho TCP + Dados), sendo interessante para se testar tentativas de sobrecarga em uma aplicação ou serviço. Uma regra também pode conter referências relacionadas ao ataque, como por exemplo, o código CVE da vulnerabilidade explorada ou uma URL (*Uniform Resource Locators*).

A partir das regras, o **Snort** gera alertas de segurança. Esses alertas é que foram utilizados para validar a OntoSec. O trecho abaixo mostra um exemplo de alerta relacionado ao protocolo *"ICMP"* e sua regra correspondente[9].

```
ALERTA GERADO
[**] [1:1917:6] SCAN UPnP service discover attempt [**]
[Classification: Detection of a Network Scan] [Priority: 3]
01/20-14:59:17.809869 XXX.XXX.XXX.206:8008 -> XXX.XXX.XXX.250:1900
UDP TTL:4 TOS:0x0 ID:47589 IpLen:20 DgmLen:129
Len: 101

REGRA CORRESPONDENTE
alert udp $EXTERNAL_NET any -> $HOME_NET 1900 (msg:"SCAN UPnP service discover attempt";
content:"M-SEARCH "; depth:9; content:"ssdp|3A|discover"; classtype:network-scan; sid:1917; rev:6;)
```

2.3.3 Firewalls

Um *firewall* é uma coleção de hardware e software projetados para examinar o tráfego e os serviços requisitados em uma rede, impedindo acessos externos não autorizados (Oppliger, 1996). Chapman e Zwick (1995) ainda definem *firewall* como uma barreira de proteção para prevenir que os perigos da Internet possam se espalhar em uma rede interna. Além disso, um *firewall* serve a vários propósitos:

- Restringe o acesso de agentes externos a um ambiente controlado.
- Previne que intrusos se aproximem dos sistemas de segurança de uma rede interna.
- Restringe o acesso de agentes internos a um ambiente controlado à Internet.

Em geral, um *firewall* pode ser um roteador, um computador que executa um filtro de pacotes, um servidor *proxy*, ou ainda uma combinação de roteadores, computadores

[8]Comando que permite traçar a rota de um pacote na rede.
[9]Parte dos endereços IPs foi suprimida para preservar a identidade das redes.

e software apropriados. Seu principal propósito é eliminar o fluxo de pacotes ou de requisições que não se encaixem nas regras previamente estabelecidas pela política de segurança de uma organização (Ray e Anonymous, 2001).

Normalmente um *firewall* é instalado no ponto onde a rede interna se conecta com a Internet. Todo o fluxo de dados vindo da Internet ou indo para a Internet passa pelo *firewall*, fazendo, assim, com que ele funcione como um filtro. A Figura 2.2 ilustra esse esquema. Um *firewall* pode também ser implementado dentro da rede interna, por exemplo, para isolar os serviços críticos da rede (servidores de arquivos, servidores de *e-mails*, servidores Web).

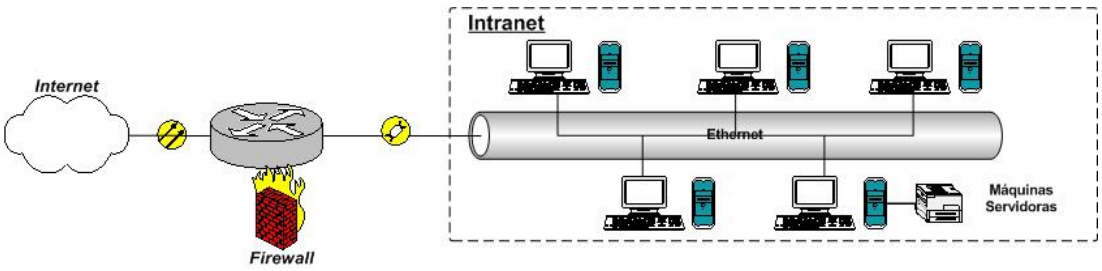

Figura 2.2: Localização típica de um *firewall* em uma rede local.

Assim como um IDS, um *firewall* também possui vantagens e limitações. Como vantagens têm-se (Chapman e Zwick, 1995):

- Concentrar o tráfego da rede, tornando o gerenciamento da segurança menos oneroso.
- Garantir que as políticas de segurança estabelecidas sejam cumpridas.
- Coletar informações sobre o tráfego da rede, registrando o que entra o que sai da rede interna.
- Limitar a exposição, evitando, por exemplo, que um problema de alguma sub-rede possa afetar toda a rede.

Por outro lado, um *firewall* também apresenta algumas limitações, tais como:

- Se o *firewall* está implementado como a Figura 2.2 mostra, ele não pode proteger de ataques maliciosos que ocorram dentro da rede interna.
- O *firewall* não pode controlar aquelas conexões que não passam por ele, por exemplo, se a rede interna permite conexão por *modem* e esse *modem* está atrás do *firewall*.

- O *firewall*, geralmente, não protege contra vírus, pois isso implica verificar o conteúdo dos pacotes na rede e não somente endereços (ou portas) de destino e de origem.

Sendo tipicamente a primeira barreira de proteção contra invasões, os *firewalls* devem ter sua configuração cuidadosamente implementada e testada antes que conexões sejam estabelecidas entre a rede interna (intranet) e a Internet. É importante, portanto, isolar os servidores de acessos externos.

2.4 Formatos para representar Dados de Segurança

Além dos desafios já apresentados neste capítulo relacionados à segurança computacional, ataques, invasões, implantações de mecanismos e ferramentas de proteção, um outro desafio pertinente é o gerenciamento dos dados e informações geradas por esses diversos mecanismos e ferramentas.

Em sistemas de segurança computacional, o administrador tem em mãos uma grande quantidade de dados e informações que devem ser gerenciados e analisados a fim de tomar uma decisão caso o sistema seja invadido. Esses dados podem ser de fontes diretamente relacionadas à segurança e/ou de fontes que podem fornecer dados que auxiliam na elaboração de informações que podem caracterizar uma quebra de segurança.

Diferentes institutos de pesquisas e pesquisadores têm realizado esforços no sentido de catalogar e classificar dados e informações relacionados à segurança computacional. O projeto CVE apresenta um padrão para nomenclatura de vulnerabilidades e exposições que facilita a identificação de uma mesma vulnerabilidade em diferentes ferramentas (Mann e Christey, 1999; Martin, 2001). O CERT, além de informações relacionadas a vulnerabilidades, também armazena informações de incidentes de segurança, alertas de vírus, soluções para problemas de segurança, entre outras.

O IDWG do IETF desenvolveu o padrão IDMEF (Curry et al., 2005) para troca de mensagens entre IDSs. O IDWG provê uma maneira padronizada para que IDSs relatem eventos considerados suspeitos, de maneira a possibilitar a interoperabilidade entre diferentes IDSs, permitindo ao usuário combinar o uso de diferentes ferramentas.

Howard (1997), em seu trabalho de doutorado, definiu uma taxonomia de segurança baseada em ataques. Essa taxonomia inclui principalmente listas de conceitos e de categorias. Essa proposta de taxonomia foi desenvolvida a partir das ações realizadas durante um ataque.

A seguir, uma descrição dessas pesquisas é apresentada.

2.4.1 O Projeto CVE

Segundo Martin (2001), organizações que descobrem uma vulnerabilidade, geralmente agem como se fossem as únicas fontes de informação a respeito daquela vulnerabilidade e utilizam sua própria abordagem para quantificar, nomear, descrever e compartilhar a informação. Essa postura torna difícil o gerenciamento da informação existente com relação às muitas vulnerabilidades encontradas nos diferentes produtos de software. A Tabela 2.1 ilustra esse problema, na qual uma mesma vulnerabilidade foi relatada (em 1998) de diferentes maneiras por diferentes organizações. Essa vulnerabilidade foi descoberta em um *script* que consiste em uma agenda, conhecida como PHF. Explorando essa falha, o invasor poderia executar códigos arbitrários no servidor Web e, eventualmente, modificar seu conteúdo.

Tabela 2.1: Mesma vulnerabilidade com descrições diferentes (Martin, 2001; Martin et al., 2003).

Organização	**Descrição**
AXENT	*phf CGI allows remote command execution*
BindView	*#107 - cgi-phf*
BugTraq	*PHF Attacks - Fun and games for the whole family*
CERIAS	*http_escshellcmd*
CERT	*CA-96.06.cgi_example_code*
CISCO	*http - cgi-phf*
CyberSafe	*Network: http 'phf' Attack*
DARPA	*0x00000025 = http PHF attack*
IBM ERS	*ERS-SVA-E01-1996:002.1*
ISS	*http - cgi-phf*
Symantec	*#180 http Server CGI example code compromises http server*
Security Focus	*#629 - phf Remote Command Execution Vulnerability*

No sentido de facilitar o gerenciamento do compartilhamento das informações relacionadas às vulnerabilidades, o Instituto Mitre criou o projeto CVE em 1999. Esse projeto tem como principal objetivo criar uma base de dados padronizada de alertas de vulnerabilidades. Atualmente, 71 organizações (entre elas: CERT/CC, CERIAS, SANS, CISCO, SYMANTEC, IBM, SECURITYFOCUS) participam desse projeto e atualizam constantemente a base de alertas, chamada lista CVE (*CVE list*). Essa lista tem a intenção apenas de nomear de maneira única as vulnerabilidades, e não de representá-las ou classificá-las.

Essa base centralizada possui as seguintes características:

- Enumerar as vulnerabilidades conhecidas.

- Atribuir um nome padrão e único para cada vulnerabilidade.
- Ser independente das diferentes perspectivas em que a vulnerabilidade ocorre.
- Ser aberta e compartilhada.

A manutenção da lista é resultado de uma discussão aberta e colaborativa entre o corpo editorial do CVE, que é composto por especialistas em segurança de organizações comerciais, acadêmicas e de pesquisa. Com o suporte do Instituto Mitre, o corpo editorial identifica quais vulnerabilidades podem ser incluídas ou não (candidatas CVE - *CVE candidates*) na lista CVE.

Até outubro de 2005, cada vulnerabilidade candidata recebia uma identificação atribuída pelo CNA (*Candidate Numbering Authority*). Essa identificação era semelhante à identificação de uma vulnerabilidade já incluída na lista. No entanto, ao invés de CVE na identificação da vulnerabilidade, tinha-se CNA (CNA-1999-0002). A partir de outubro de 2005, todas as vulnerabilidades candidatas têm o prefixo CVE, no entanto, são classificadas como *candidate*.

Cada entrada na lista oficial CVE inclui as seguintes informações:

- Número de identificação CVE (por exemplo, "CVE-2004-0167").
- Indicação se já faz parte da lista CVE ou não (status "*entry*" ou "*candidate*").
- Descrição da vulnerabilidade.
- Qualquer referência pertinente à vulnerabilidade (por exemplo, uma URL que descreve a vulnerabilidade, ou a identificação equivalente de alguma organização).

A Tabela 2.2 mostra alguns exemplos de entradas da lista oficial CVE[10]. Os nomes são concatenados pelo ano de descoberta da vulnerabilidade e ordenados de maneira seqüencial. As descrições não apresentam uma estrutura padrão. Assim, o nome "CVE-2004-0089" indica somente a octogésima nona vulnerabilidade registrada em 2004.

A simplicidade do projeto CVE oferece diversas vantagens, tais como: criar uma base de alertas de vulnerabilidades compartilhada e mantida por diversas organizações, estabelecer uma compatibilidade entre diferentes ferramentas de segurança que utilizam o padrão CVE. Porém, a iniciativa não oferece avanços na correlação entre vulnerabilidades e na extração de valor semântico, já que a simplicidade do projeto privou-o da possibilidade de relacionar maiores informações referentes às vulnerabilidades catalogadas. Além disso,

[10]Versões da lista CVE podem ser encontradas em `http://www.cve.mitre.org/cve/versions/`.

Tabela 2.2: Exemplo de vulnerabilidades cadastradas na lista oficial CVE.

Código CVE	Status	Descrição	Referência
CVE-2004-0167	*Entry*	*DiskArbitration in MacOS X 10.2.8 and 10.3.2 does not properly initialize writeable removable media*	*CERT-VN:VU#578886*
CVE-2004-0122	*Entry*	*Microsoft MSN Messenger 6.0 and 6.1 does not properly handle certain requests, which allows remote attackers to read arbitrary files*	*CERT-VN:VU#688094*
CVE-2004-0089	*Entry*	*Buffer overflow in TruBlueEnvironment in MacOS X 10.3.x and 10.2.x allows local users to gain privileges via a long environment variable*	*APPLE:APPLE-SA-2004-01-26*

o uso de nomes seqüenciais dificulta a correlação entre diferentes entradas. Por exemplo, é difícil prever automaticamente se existe alguma correlação entre as vulnerabilidades "CVE-2001-0012" e "CVE-2002-0301". Além disso, o fato das descrições não seguirem um padrão de estrutura dificulta a extração do valor semântico dos dados, ficando, assim, a cargo dos administradores analisarem essas descrições (Brandão, 2004).

2.4.2 CERT/CC

Em novembro de 1988, um aluno de pós-graduação implementou e espalhou pela Internet um programa que se autoduplicava, chamado *Internet Worm*. Esse programa explorava diversos problemas do sistema operacional UNIX com o intuito de invadir computadores ao longo da rede de computadores. Inúmeros computadores foram infectados e tiveram sua capacidade de processamento absorvida pelas múltiplas cópias do programa *worm*. Com o intuito de eliminar esse *Internet Worm*, uma força tarefa (*response team*) com peritos do MIT (*Massachusetts Institute of Technology*), Berkeley, Purdue, entre outros, foi criada. Esse grupo realizou a engenharia reversa do código do *worm* e procedimentos foram criados para erradicá-lo. A partir desse incidente, a DARPA (*Defense Advanced Research Projects Agency*) decidiu institucionalizar uma força tarefa de emergências na Internet, criando assim o CERT/CC.

O CERT foi criado como parte do Instituto de Engenharia de Software da Universidade de Carnegie Mellon nos EUA, com o intuito de prover uma organização capaz de coordenar respostas a incidentes de segurança na Internet.

O CERT também mantém uma base de dados com as vulnerabilidades conhecidas na Internet. Essa base também possui as correções relacionadas a cada vulnerabilidade. Ao contrário do projeto CVE, as informações mantidas pelo CERT são restritas e confidenciais, somente as partes interessadas têm acesso a essas informações. Somente as informações relacionadas às características gerais da vulnerabilidade e os detalhes específicos de como eliminar essa vulnerabilidade são públicas. Essa política é adotada a fim de impedir um ataque a tal vulnerabilidade. A mesma política é adotada com incidentes em páginas na Internet. Somente o *website* atacado recebe todas as informações. Caso o *website* autorize, as informações se tornam públicas.

Segundo o CERT, um incidente de segurança é definido como **um ato de violar uma política de segurança**. Essa definição recai sobre a existência de uma política de segurança, e, como já discutido, uma política de segurança varia de organização para organização. Assim, O CERT caracteriza as seguintes atividades como tipos de violações largamente reconhecidas de uma política de segurança típica:

- Tentativas (com sucesso ou não) de se ganhar acesso não autorizado a um sistema ou a um conjunto de dados.
- Negação de Serviço.
- Uso não autorizado de um sistema para processamento ou armazenamento de dados.
- Mudanças em sistemas de hardware, *firmware*, ou características de software sem conhecimento ou autorização dos proprietários.

Tal como no projeto CVE, o CERT não realiza uma correlação entre as diferentes vulnerabilidades e incidentes, impossibilitando, assim, identificar uma correlação entre vulnerabilidades e incidentes. Além disso, a maneira como os dados são armazenados não facilita uma análise semântica das informações, como por exemplo, identificar quais são as consequências para o sistema de um determinado incidente ou de uma vulnerabilidade.

2.4.3 O padrão IDMEF

O padrão IDMEF foi criado pelo grupo IDWG (do IETF) para facilitar a troca de mensagens entre IDSs.

Para a representação das mensagens, o IDWG optou pela utilização da linguagem XML (*eXtensible Markup Language*) (Beckett, 2004; Bray et al., 2004). O modelo divide os alertas em classes e subclasses. A Figura 2.3 apresenta visão geral do modelo. Todas as

mensagens trocadas pertencem à classe **IDMEF-Message**, que se divide nas subclasses **Alert** e **Heartbeat**. A primeira é utilizada para enviar informações sobre alertas de invasão, disparadas no momento da detecção. Já a segunda classe é usada para envio de mensagens de *status*, disparadas em intervalos regulares.

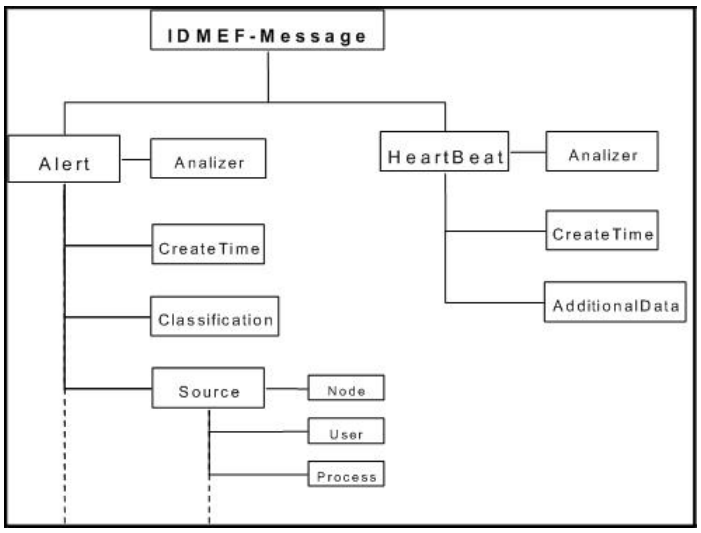

Figura 2.3: Parte do modelo de dados do padrão IDMEF (Curry et al., 2005).

Como mostra a Figura 2.3, a subclasse **Alert** tem os seguintes atributos relacionados à atividade reportada:

- *Analyzer*: Agente (ferramenta de segurança) que gerou o alerta.
- *CreateTime*: Hora de criação do alerta.
- *DetecTime*: Hora de detecção da atividade suspeita. Geralmente tem o mesmo valor que o atributo anterior.
- *AnalyzerTime*: Hora corrente no agente gerador do alerta.
- *Source*: Origem do ataque.
- *Target*: Alvo do ataque.
- *Classification*: Nome do alerta ou outra forma de identificá-lo. Pode-se, por exemplo, utilizar o padrão CVE.
- *Assessment*: Estima o impacto do ataque, contra-medidas adotadas e o grau de sigilo da atividade ocorrida.

- *AdditionalData*: Informações extras que não se encaixem na estrutura modelo.

A subclasse **Heartbeat**[11] utiliza os seguintes atributos:

- *Analyzer*: Agente gerador da mensagem.
- *CreateTime*: Horário de criação da mensagem.
- *AditionalData*: Informações extras que não se encaixam na estrutura do modelo.

O modelo é representado a partir de um documento DTD (*Document Type Definition*)[12], como mostra a Figura 2.4. No exemplo, um relatório de ataque a uma vulnerabilidade presente no programa de agenda, conhecido como PHF, é mostrado.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFC XXXX IDMEF v1.0//EN"
  "idmef-message.dtd">
<IDMEF-Message version="1.0">
  <Alert ident="abc123456789">
    <Analyzer analyzerid="bc-sensor01">
      <Node category="dns">
        <name>sensor.example.com</name>
      </Node>
    </Analyzer>
    <CreateTime ntpstamp="0xbc71e980.0x00000000">
      2000-03-09T08:12:32-01:00
    </CreateTime>
    <Source ident="abc123">
      <Node ident="abc123-001">
        <Address ident="abc123-002" category="ipv4-addr">
          <address>192.0.2.200</address>
        </Address>
      </Node>
      <Service ident="abc123-003">
        <port>21534</port>
      </Service>
    </Source>
    <Target ident="xyz789">
      <Node ident="xyz789-001" category="dns">
        <name>www.example.com</name>
        <Address ident="xyz789-002" category="ipv4-addr">
          <address>192.0.2.100</address>
        </Address>
      </Node>
      <Service>
        <port>8080</port>
        <WebService>
          <url>
            http://www.example.com/cgi-bin/phf?/etc/group
          </url>
          <cgi>/cgi-bin/phf</cgi>
          <http-method>GET</http-method>
        </WebService>
      </Service>
    </Target>
    <Classification origin="bugtraqid">
      <name>629</name>
      <url>http://www.securityfocus.com</url>
    </Classification>
  </Alert>
</IDMEF-Message>
```

Figura 2.4: Exemplo da representação de um ataque com o padrão IDMEF.

O uso de XML acarreta em algumas limitações ao IDMEF. Por exemplo, XML não conta com o conceito de herança e não há representação de correlação entre mensagens.

[11] Similar ao batimento cardíaco, de acordo com a tradução literal.

[12] `http://www.w3.org/TR/html4/sgml/dtd.html`.

Além disso, o significado do conteúdo XML é baseado na interpretação dos nomes das marcações, o que não permite a um agente de software realizar inferências a partir de seu conteúdo.

2.4.4 A Taxonomia de Howard

Howard (1997) definiu uma taxonomia baseada em ataques. Um ataque é uma tentativa de acesso não autorizado. Um atacante tenta estabelecer um *link* com um determinado objetivo. Esse *link* é estabelecido por meio de uma seqüência operacional de ferramentas, vulnerabilidades, ações, alvos e resultados que conectam o atacante aos seus objetivos. A Figura 2.5 ilustra essa seqüência operacional, bem como os principais conceitos da taxonomia. Alguns dos conceitos da OntoSec foram herdados dessa taxonomia.

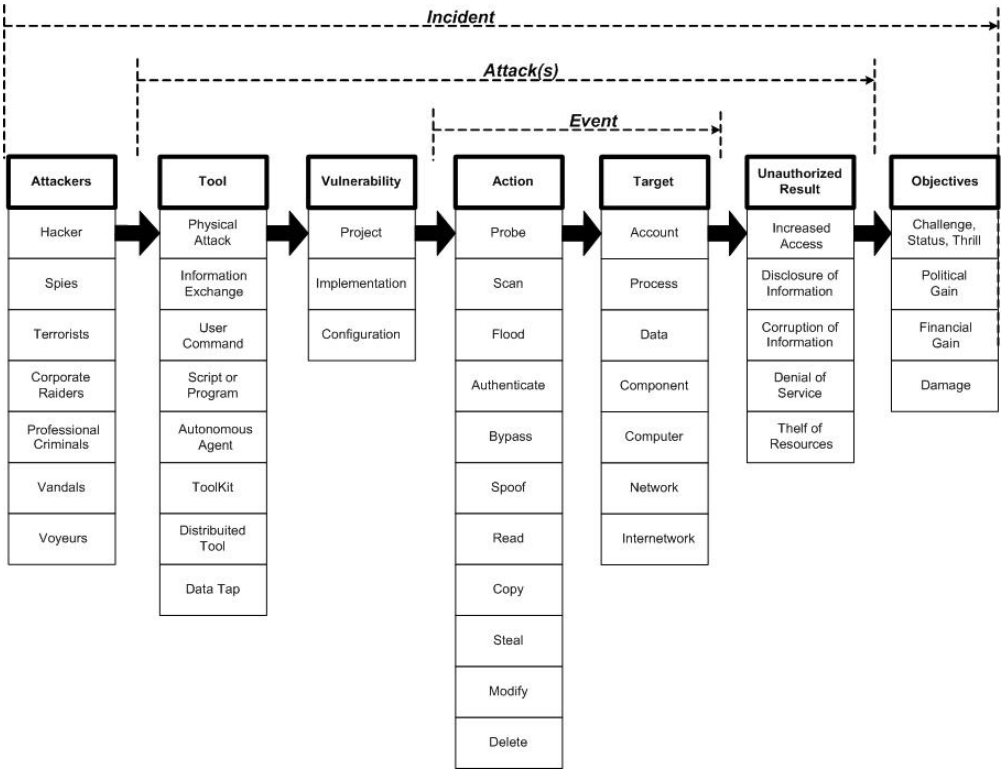

Figura 2.5: Taxonomia de incidentes de segurança definida por Howard e Longstaff (1998).

A taxonomia procura rastrear o caminho que um **atacante** pode percorrer a fim de atingir seus **objetivos**, caracterizando um *incidente*. Como pode ser observado pela Figura 2.5, uma **ação** direcionada a um determinado **alvo** com o objetivo de modificar o estado desse alvo caracteriza em *evento*. Um *ataque* é caracterizado pela ocorrência de um *evento* que utiliza uma **ferramenta** para explorar uma **vulnerabilidade** e al-

cançar um **acesso** não autorizado (**resultado**). Nesse sentido, os esforços da segurança computacional têm como meta observar e conhecer cada um dos elementos descritos pela taxonomia.

A seguir, os principais conceitos da taxonomia de incidentes de segurança são descritos.

Attackers. Atacantes são caracterizados pelas pessoas que realizam ataques. Eles podem ser identificados por quem são, onde estão e quais são suas habilidades. Os atacantes podem ser divididos nas categorias a seguir, e, dependendo da categoria, os objetivos podem ser diferentes.

- ***Hackers***: Quebram um sistema pelo desafio e para obter acesso (***objective***:***challenge, status, thrill***).
- ***Spies***: Quebram um sistema com intuito de ganhos políticos (***objective***:***political gain***).
- ***Terrorists***: Quebram um sistema com o intuito de ganhos políticos e de causar medo (***objective***:***political gain***).
- ***Corporate raiders***: Empregados internos que quebram um sistema com intuito de ganhos financeiros (***objective***:***financial gain***).
- ***Professional Criminals***: Quebram um sistema com intuito de ganhos financeiros (***objective***:***financial gain***).
- ***Voyeurs***: Quebram um sistema para causar danos (***objective***:***damage***).

Tool. As ferramentas são os meios pelos quais os atacantes obtêm acessos. Existem, atualmente, inúmeras dessas ferramentas, que estão, na maioria das vezes, disponíveis na Internet. Howard e Longstaff (1998) dividem as ferramentas nas seguintes categorias:

- ***Physical Attack***: Caracteriza um roubo ou um estrago físico de componentes do sistema, como por exemplo computadores, roteadores, energia elétrica, etc.
- ***Information Exchange***: Obter informações do sistema por meio de outros atacantes ou de pessoas envolvidas no sistema (engenharia social).
- ***User Command***: Comandos executados em alguma interface gráfica ou linha de comando. Um exemplo é abrir uma sessão de *telnet* em um computador alvo utilizando uma conta de administrador ou de um usuário comum.

- ***Scripts*** ou ***Programs***: O atacante utiliza *scripts* ou programas para explorar vulnerabilidades. Um exemplo é o cavalo de tróia (*trojan horse*) que aparentemente é um programa normal, mas esconde um código malicioso.
- ***Autonomous Agents***: O atacante utiliza programas ou fragmentos de programas para explorar vulnerabilidades. Ao contrário da categoria acima, esses agentes são capazes de escolher seus alvos. Exemplos são os vírus e os *worms*. Os *worms* são programas que, depois que são iniciados, se autoduplicam e se espalham, causando problemas sem que haja intervenção do usuário.
- ***Toolkits***: São pacotes de software compostos por diferentes ferramentas, tais como: *scripts*, programas, agentes autônomos, entre outras. Um exemplo é o *rootkit*, que contém um *sniffer*[13] e um cavalo de tróia que podem ser utilizados para esconder atividades não autorizadas ou instalar *backdoors*[14].
- ***Distributed Tool***: Utilizada para atacar um computador a partir de vários outros computadores. Determinar um ataque realizado com esse tipo de ferramenta é uma tarefa difícil, pois o ataque chega de várias fontes. Uma característica interessante é a sincronização do ataque. Ataques DDoS (*Distributed Denial of Service*) são o resultado da união de negação de serviço e intrusão distribuída. Eles podem ser definidos como ataques DoS diferentes partindo de várias origens, disparados simultânea e coordenadamente sobre um ou mais sistemas.
- ***Data Tap***: Nesse caso, o atacante utiliza o meio físico para realizar o ataque. Esse tipo de ferramenta não é muito comum, mas faz parte da taxonomia para garantir completude.

É importante ressaltar que no início dos anos 80, os intrusos eram normalmente pessoas com alto conhecimento tecnológico que criavam seus próprios métodos para invadir sistemas. O uso de ferramentas automáticas e *scripts* era exceção e não a regra. Atualmente, qualquer pessoa pode atacar uma rede, devido à facilidade com que se conseguem tais ferramentas e *scripts* (Allen et al., 2000). Enquanto os intrusos mais experientes estão se tornando mais eficientes e espertos, o que pode ser comprovado pelo crescimento da sofisticação nos tipos de ataques (vide Figura 2.6), o conhecimento requerido por parte de intrusos novatos para copiar e iniciar alguns desses métodos de ataques está decaindo.

[13]Um *sniffer* é um programa que captura pacotes que estão trafegando em uma rede.
[14]Um *backdoor* é caracterizado por uma brecha que é aberta no sistema pelo atacante.

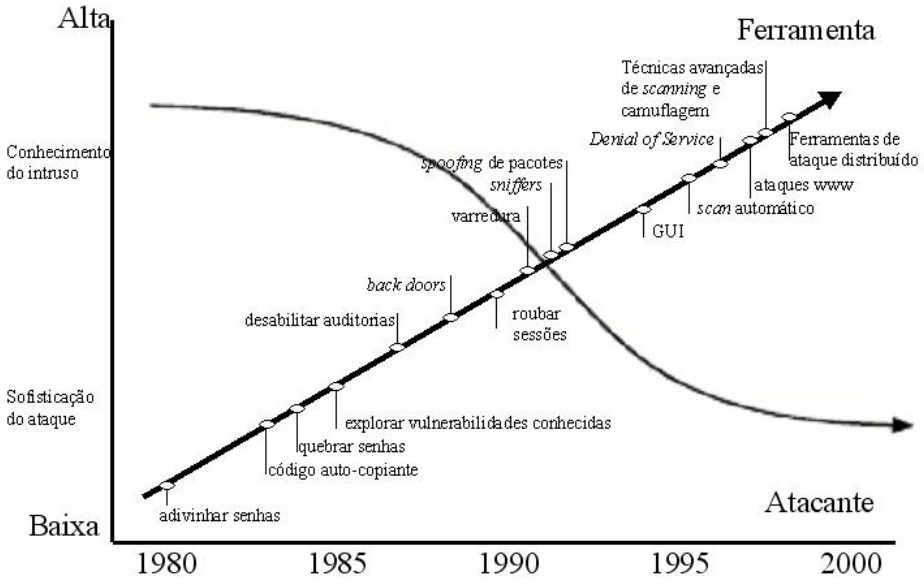

Figura 2.6: Evolução das ferramentas de ataque X Conhecimento dos atacantes (Allen et al., 2000).

Vulnerability. Os tipos de acessos podem ser autorizados ou não autorizados. Esses acessos são realizados em processos, arquivos ou dados entre processos que se comunicam. É possível encontrar na literatura *probe* como uma tentativa de ganhar acesso não autorizado a um sistema ou de descobrir informações sobre o sistema, e *scan* como um grande número de *probes* feito a partir de uma ferramenta automatizada. Os *scans* podem, algumas vezes, ser resultado de um sistema mal configurado ou de alguma vulnerabilidade. Para obter um acesso desejado, o atacante explora, geralmente, essa vulnerabilidade. Uma vulnerabilidade pode estar relacionada à implementação, ao projeto, ou à configuração.

Vulnerabilidades de implementação são as mais comuns e estão relacionadas a erros de programação. Geralmente, vulnerabilidades de projeto levam a vulnerabilidades de implementação. Por exemplo, um erro de projeto no programa `Sendmail` possibilita *spams*[15]. Vulnerabilidades de configuração também são bastante comuns e estão relacionadas, principalmente, às configurações *default* de um software. Exemplos desse tipo de vulnerabilidade são: (i) criação de determinadas contas no sistema com todos os tipos de acessos; (ii) instalação de sistemas operacionais cuja configuração *default* é deixar todas as portas abertas.

[15]Um *spam* é caracterizado por mensagens não solicitadas, por envio abusivo de correio eletrônico não solicitado, geralmente em grande quantidade. Um *spam*, aparentemente, não causa perdas diretas aos sistemas, no entanto, quando bem planejado pode levar a um DoS nos serviços de *e-mail*.

Os acessos ilegais a um sistema, segundo Howard e Longstaff (1998), podem ser evitados de duas maneiras. A primeira maneira é por meio de um rigoroso programa de descoberta e eliminação das vulnerabilidades. Os administradores devem, portanto, estar atentos aos *patches* de atualização. Quanto mais tempo se leva para descobrir a vulnerabilidade e aplicar tais *patches*, maior a probabilidade de se ter o sistema comprometido (atacado). A segunda maneira é garantir que o acesso controlado a arquivos e processos esteja implementado apropriadamente. Essa garantia inclui desde o controle de senhas ditas **fortes** (com uso de caracteres especiais, letras e números) e criptografia de senhas, até correção de erros de configuração em permissões de arquivos.

Unauthorized Results. Após obter acesso, o atacante está livre para explorar o local acessado, podendo alterar e obter informações, negar ou bloquear serviços, ou utilizar os serviços disponíveis. Os resultados de um ataque podem ser:

- ***Increased Access***: Caracteriza acessos não autorizados ao sistema.
- ***Disclosure of Information***: Disseminação da informação para pessoas não autorizadas.
- ***Corruption of Information***: Alteração não autorizada de arquivos ou de dados transmitidos pela Internet.
- ***Thelf of Resources***: Uso não autorizado de serviços oferecidos pelo sistema.
- ***Denial of Service***: Degradação ou negação de serviços aos usuários autorizados.

2.5 Considerações Finais

Para cada forma de ataque, existem diferentes e inúmeras ferramentas disponíveis que podem ser utilizadas. Por outro lado, existe uma grande quantidade de ferramentas e métodos que auxiliam na tarefa de proteger os sistemas computacionais. Esses métodos, ferramentas e formas de ataques geram uma grande quantidade de informações acerca do ambiente que se deseja proteger. Além disso, existe um grande número de alertas de incidentes de segurança que precisam ser analisados, compreendidos e correlacionados, tornando onerosa a tarefa dos administradores.

Para definir a segurança de um sistema computacional é importante compreender todo o ambiente em que o sistema está inserido e não somente as tecnologias particulares. A segurança em si é um sistema interconectado (uma cadeia), no qual diferentes componentes

interagem trocando informações. Se uma das partes está fraca (pouco segura) todo o sistema está comprometido. Assim, segurança computacional é um processo e não um produto. Sendo um processo, a segurança computacional deve permear todo o sistema: seus componentes, tecnologias, níveis de complexidade e interações (Schneier, 2000). Vale ressaltar que mesmo com a constante evolução das tecnologias de proteção, não existem, infelizmente, sistemas 100% seguros.

Diferentes ferramentas podem e devem ser combinadas para garantir uma melhor segurança aos sistemas computacionais. Essas ferramentas normalmente armazenam e geram informações de segurança em formatos próprios, dificultando a correlação e o entendimento dos incidentes de segurança. Nesse caso, o administrador deve, muitas vezes, utilizar seu conhecimento tácito para inferir e correlacionar incidentes aparentemente diferentes.

Dentro desse contexto, faz-se necessário definir um formato único para armazenar e gerar informações de segurança com os objetivos de: (i) compartilhar conhecimento comum do domínio de incidentes de segurança; e (ii) facilitar a correlação entre incidentes de segurança, possibilitando que agentes de software possam realizar inferências sobre os incidentes e possam gerar conhecimento dedutivo.

Ontologias têm sido propostas como uma alternativa para criar modelos únicos para representar conceitos e relacionamentos de um determinado domínio de conhecimento. Nesse sentido, o próximo capítulo apresenta como ontologias podem ser utilizadas para representar informação e conhecimento, facilitar o gerenciamento e o compartilhamento.

Capítulo

3

Ontologias

3.1 Considerações Iniciais

Os sistemas de informações atuais têm que ser capazes de resolver a interoperabilidade semântica existente nas grandes bases de dados, nas quais um fato pode ter mais de uma descrição. Os sistemas devem entender o modelo que o usuário faz do mundo e seus significados e entender também os modelos embutidos nas diversas fontes de informação (Fonseca et al., 2000). Entendendo os modelos, as tarefas de gerenciar e gerar conhecimento se tornam mais fáceis e menos custosas.

Nos últimos anos, pesquisadores têm proposto a aplicação de diferentes perspectivas ou tecnologias a fim de facilitar as tarefas de gerar e gerenciar conhecimento. A Ontologia é uma dessas perspectivas. O conceito de Ontologia surgiu originalmente na Filosofia e tem sido utilizada nas ciências da computação desde o final dos anos 70, quando foi introduzida por John McCarthyno. Segundo a Filosofia, ontologia é definida como *"a explicação sistemática da existência - o estudo do ser"*.

Na Filosofia, uma ontologia é utilizada para descrever as coisas naturais do mundo e a existência dos seres em si. Em outras áreas, como Computação, Engenharia, Medicina, ontologias podem ser utilizadas por aplicações que necessitam compartilhar conhecimento. Para tanto, utiliza-se um vocabulário específico e um conjunto de axiomas lógicos, que não apenas fornecem uma hierarquia sintática e semântica pretendida aos conceitos desse vocabulário, mas que também restringem a interpretação e o uso desses conceitos. Além

disso, uma ontologia permite uma interpretação automática dos conceitos e relacionamentos de um domínio de conhecimento (Noy e McGuinness, 2001).

Diversos pesquisadores têm estudado o uso de ontologias em diversos domínios de conhecimento, tais como: comércio eletrônico (Deepsia, 2001; Eulália et al., 2002; Herrera et al., 2002), química (López, 1996; López et al., 1999), segurança (Brandão, 2004; Brandão et al., 2004; Chen et al., 2002; Filman e Linden, 1996; Martimiano et al., 2004; Martimiano e Moreira, 2006a; Raskin et al., 2001; Schumacher, 2003; Undercoffer et al., 2003), entre outros (Everett et al., 2002; Gruninger e Fox, 1995; Gruninger e Lee, 2002; Holsapple e Joshi, 2002; Lima et al., 2002; Pacheco e Kern, 2001; Uschold e Gruninger, 1996; Zuñiga, 2001), mostrando o potencial dessa abordagem de representação de conhecimento.

Na definição de uma ontologia para vinhos, por exemplo, deve-se incluir a especificação dos seguintes conceitos e relacionamentos: *vinho*, *cor do vinho*, *tinto*, *branco*, *rose*, *seco*, *suave*, *demiseco*, *ano safra*, *vinícola*, *produz vinho*, *é produzido* entre outros. Na ontologia, esses conceitos são representados por classes, ou por atributos ou por relacionamentos. Além disso, a ontologia também define uma hierarquia de classes, onde subclasses herdam características da superclasse. Por exemplo, a classe **Vinho** pode ser especializada em **VinhoTinto**, **VinhoRose** e **VinhoBranco**. Pode-se ainda restringir que vinhos têm um único ***ano_safra***, cujo valor não pode ser negativo. Ainda, pode-se especificar que uma vinícola (**Vinicola**) produz (*produz_vinho*) vários vinhos utilizando restrição de cardinalidade 1:N. Pode-se ainda especificar que um vinho é produzido (*e_produzido*) por várias vinícolas com cardinalidade de 1:N. É possível também definir que um relacionamento é inverso de outro relacionamento. Nessa ontologia de vinhos, por exemplo, se um vinho é produzido por uma determinada vinícola, essa vinícola produz aquele vinho. Assim, os relacionamentos *produz_vinho* e *e_produzido* são inversos.

A seguir, neste capítulo, os conceitos e benefícios de ontologia são descritos, incluindo a engenharia de ontologias, algumas linguagens de representação e ferramentas para editar e avaliar ontologias.

3.2 O Conceito de Ontologia

Segundo Gruber (1993), uma ontologia é uma **"especificação formal e explícita de uma abstração, uma visão simplificada de um domínio de conhecimento"**. Para Noy e McGuinness (2001), uma ontologia é uma **"descrição formal e explicíta dos conceitos de um domínio de conhecimento, das suas propriedades (atributos e relacionamentos) e restrições"**. Os relacionamentos entre os conceitos podem

ser hierárquicos (relações do tipo *isA*), representando generalização (ou hiperonímia) e especialização (ou hiponímia), ou não hierárquicos, que são aqueles que representam eventos.

O desenvolvimento de uma ontologia se assemelha ao processo de desenvolvimento de um sistema especialista ou um sistema baseado em conhecimento. Assim como nesses sistemas, é necessário que um especialista do domínio de conhecimento acompanhe o processo de desenvolvimento da ontologia, a fim de validar os conceitos e relacionamentos que estão sendo modelados. Além do especialista, livros, artigos e trabalhos relacionados ao domínio devem ser consultados.

Independentemente do domínio representado pela ontologia e das diferenças existentes, alguns elementos são comuns (Chandrasekaran et al., 1999):

- Existem **objetos** do mundo real a serem representados. Esses objetos são traduzidos em conceitos na ontologia e esses conceitos podem ser agrupados em classes.
- Objetos têm **atributos** que podem assumir valores variados.
- Objetos podem se **relacionar** com outros objetos.
- Atributos e relacionamentos podem mudar.
- Existem **eventos** que podem ocorrer em determinados instantes.
- Existem **processos** que podem ocorrer ao longo do tempo.
- O mundo e seus objetos podem estar em diferentes **estados**.
- Eventos podem **causar** outros eventos ou estados como **efeitos**.
- Objetos podem ser decompostos ou fazerem parte de outros objetos (**agregação**), ou podem ser especializados em outros objetos (**especialização**).

Cada um desses elementos é instanciado dependendo única e exclusivamente do domínio que a ontologia está representando.

3.2.1 O Uso de Ontologias

O grau de formalidade pelo qual um vocabulário é criado varia. Segundo Uschold e Gruninger (1996), esse grau de formalidade pode ser:

- **Altamente Informal**: Vocabulário é expresso unicamente em língua natural.

- **Semi-informal**: Vocabulário é expresso em alguma língua natural restrita e estruturada, reduzindo a ambigüidade.

- **Semi-formal**: Vocabulário é expresso em uma linguagem formalmente definida.

- **Rigorosamente Formal**: Vocabulário é expresso cuidadosamente em conceitos com semântica formal, teoremas e provas para garantir completude.

Independentemente de seu grau de formalidade, para que uma ontologia seja útil, é preciso que ela seja compreendida por seus usuários. Complexas equações matemáticas e grandes árvores de elementos inter-relacionados não motivam usuários a extraírem informações e conhecimento úteis de uma ontologia.

Para Bézivin (1998), ontologias são consideradas uma evolução perante vocabulários, dicionários de dados, esquemas de bancos de dados e taxonomias. Um vocabulário é um conjunto de palavras sem caráter universal e formal, utilizado apenas em um contexto específico. Um dicionário de dados é um vocabulário técnico utilizado no contexto específico de uma aplicação. Um esquema de banco de dados é orientado a uma determinada aplicação que define as entidades, seus atributos e relacionamentos. Uma taxonomia é uma hierarquia de conceitos baseada em relacionamentos de especialização (isA) e composição ($partOf$). Já uma ontologia compreende o conhecimento de um domínio particular sob a forma de conceitos e relacionamentos, ou seja, é orientada a um domínio.

O desenvolvimento de uma ontologia geralmente começa com o uso de uma taxonomia. Taxonomias bem estruturadas auxiliam a modelagem dos elementos de uma ontologia, dos diferentes pontos de vista da interpretação humana, além de facilitar reúso e integração. Por outro lado, taxonomias mal estruturadas causam um efeito contrário, estabelecendo modelos confusos e de difícil reúso e integração (Welty et al., 2001).

Mas, afinal, quais são os benefícios de se utilizar uma ontologia ao invés de se usar simplesmente uma taxonomia? Atualmente, diversas pesquisas têm defendido os benefícios do uso de ontologias. Segundo Noy e McGuinness (2001), os principais benefícios são:

- Compartilhamento do entendimento comum a diversas pessoas ou agentes de software de um domínio de conhecimento.

- Uso de definições explícitas e formais facilita a manutenção do conhecimento, possibilita melhor compreensão de um domínio, facilitando o reúso da ontologia (ou de parte dela) e permitindo sua extensão (inclusão de novos conceitos).

- Separar o domínio de conhecimento de uma aplicação específica, possibilitando que uma única ontologia possa ser utilizada em diferentes aplicações de um mesmo

domínio. Por exemplo, uma ontologia que descreve a tarefa de construir uma determinada peça ou equipamento de um carro, pode ser utilizada em diferentes linhas de montagens. Nesse caso, o domínio de conhecimento é "construir a peça ou equipamento".

Além dos benefícios de compartilhamento, reúso e separação entre domínio e aplicação, Uschold e Gruninger (1996) apontam ainda:

- Uma ontologia permite a **interoperabilidade** entre diferentes sistemas, nos quais diferentes usuários compartilham dados e/ou utilizam diferentes ferramentas.
- Uma representação formal torna possível a geração automática de informações e conhecimento mais **consistentes** e mais **confiáveis**.
- O conhecimento compartilhado pode ajudar no processo de identificar requisitos e definir uma **especificação** para Sistemas de Informação. Esse fato é especialmente verdadeiro quando os requisitos envolvem diferentes grupos utilizando diferentes terminologias em um mesmo domínio. Por exemplo, no domínio da cana-de-açúcar, cada usina define sua própria terminologia. Uma ontologia para cana-de-açúcar pode ser definida e compartilhada pelas várias usinas, facilitando o desenvolvimento de um sistema de apoio ao processo de moagem da cana-de-açúcar que represente os conceitos desse domínio de maneira única.

Segundo Menzies (1999), a estruturação formal e o reúso podem, no entanto, ter um alto custo. Isso porque, segundo modelos de estimativa de custo para adequação de sub-rotinas de software em novos sistemas, o tempo necessário para que um novo usuário adquira conhecimento suficiente para utilizar a sub-rotina é o tempo equivalente a reescrever 60% da sub-rotina. Esse tempo está relacionado com a necessidade do usuário em se familiarizar com as sub-rotinas.

Ainda segundo Menzies, não há garantia de que tais resultados possam ser considerados no reúso de ontologias, porém servem para alertar que a possibilidade de se utilizar uma ontologia já existente, ou parte dela, não garante por si só o aumento de produtividade. Ontologias precisam ser estudadas e compreendidas antes de serem utilizadas e o tempo de aprendizado pode ser significativo.

3.2.2 A Engenharia de Ontologias

Segundo Falbo (1998), construir uma ontologia, independentemente do domínio, é uma tarefa complexa e custosa. Gruninger e Lee (2002) identificam ainda como principais

limitações para o desenvolvimento de ontologias a dificuldade em reusar e compartilhar, conforme já observado por Menzies (1999). Essas limitações ocorrem, principalmente, devido ao fato de que as ontologias requerem um consenso de diferentes visões a respeito de um domínio de conhecimento.

Durante o desenvolvimento de uma ontologia é importante definir claramente as tarefas que devem ser realizadas. No entanto, não existe um consenso sobre uma metodologia ou outra. Normalmente, quando do desenvolvimento de uma ontologia, os desenvolvedores acabam criando sua própria metodologia. Existem diversas metodologias, mas nenhuma delas é considerada um modelo padrão de desenvolvimento. Algumas das metodologias que podem ser encontradas na literatura são: a metodologia baseada em questões de competência proposta por Gruninger e Fox (1995), as metodologias propostas por Uschold e King (1995), Uschold (1996) e Uschold e Gruninger (1996), a `Methontology` definida por (Fernández et al., 1997), a metodologia de Noy e McGuinness (2001), e a abordagem colaborativa utilizada por Holsapple e Joshi (2002). Basicamente, essas metodologias definem os seguintes passos para o desenvolvimento de uma ontologia: projeto, avaliação, validação, manutenção e uso da ontologia.

Para o desenvolvimento da OntoSec, duas metodologias foram utilizadas: a `Methontology` (Fernández et al., 1997) e a metodologia de Noy e McGuinness (2001). Essas metodologias foram utilizadas de maneira complementar. Enquanto a `Methontology` guiou o processo de desenvolvimento da OntoSec como um todo, a metodologia de Noy e McGuinness foi utilizada para definir como os conceitos, atributos e relacionamentos estão representados na OntoSec.

A `Methontology` foi utilizada por ser uma metodologia baseada tanto no padrão IEEE 1074-1995 para desenvolvimento de software (IEEE, 1996) quanto em uma metodologia para desenvolvimento de Sistemas Baseados em Conhecimento, denominada IDEAL (Gómez-Pérez et al., 1997; J., 1995). A metodologia de Noy e McGuinness foi utilizada na conceituação da OntoSec porque possui passos bem definidos e claros de como os conceitos de um domínio de conhecimento devem ser representados em uma ontologia.

Além da metodologia de Noy e McGuinness, foram definidas questões de competência para facilitar a construção do modelo conceitual. Essas questões de competências são perguntas que a ontologia deve ser capaz de responder. Assim, a partir do que se espera que a ontologia responda, fica mais fácil definir quais conceitos do domínio a ontologia deve representar e como esses conceitos se relacionam.

A seguir, essas metodologias são descritas com mais detalhes.

A Methontology

Fernández et al. (1997) propuseram uma metodologia estruturada para o desenvolvimento de ontologias: a `Methontology`. As principais fases e atividades dessa metodologia são: planejamento e especificação, conceituação, formalização, integração, implementação, manutenção, aquisição de conhecimento, documentação e avaliação. Em cada uma dessas fases, a `Methontology` gera diferentes documentos. A seguir, cada uma dessas fases é descrita.

Planejamento e Especificação: Nessas fases, as seguintes tarefas são realizadas: definição do domínio a ser representado, definição do propósito da ontologia, definição de quem são os usuários finais da ontologia, definição de quais são as tarefas que serão realizadas durante o processo de desenvolvimento da ontologia, definição dos recursos (pessoas, software, hardware) que serão necessários para a execução das tarefas e quanto tempo será necessário para cada uma delas. **Resultado**: Documento de especificação com os requisitos da ontologia.

Conceituação: Nessa fase, o conhecimento adquirido é estruturado por meio de um modelo conceitual ou de uma hierarquia descrevendo os conceitos, propriedades e restrições do domínio de conhecimento. Essa fase se caracteriza como uma das mais importantes do processo. Para definir o modelo conceitual da OntoSec, a fase de conceituação da metodologia de Noy e McGuinness (2001) foi utilizada. **Resultado**: Modelo Conceitual[1].

Formalização: Com base no modelo conceitual da fase anterior, os elementos do domínio são formalizados utilizando alguma linguagem, formal ou semiformal, de representação. A Seção 3.2.3 apresenta exemplos de linguagens de representação que têm sido utilizadas para formalizar ontologias. **Resultado**: Documento de Formalização.

Integração: Nessa fase, se necessário, a ontologia é integrada a outras ontologias ou vice-versa. **Resultado**: Documento de Integração.

Implementação: Nessa fase, a ontologia pode ser implementada utilizando uma linguagem de programação. **Resultado**: Documento de Implementação (ou código).

[1]Outros documentos podem ser gerados dependendo do domínio da ontologia que está sendo desenvolvida. Por exemplo, uma ontologia para produtos químicos pode ter uma tabela de fórmulas.

Manutenção: Essa fase também se caracteriza como uma das mais importantes. É nela que a ontologia desenvolvida evolui ou é modificada para se adequar a alguma mudança no domínio de conhecimento.

Aquisição de Conhecimento: A atividade de aquisição de conhecimento se concentra nos passos iniciais do processo (planejamento, especificação e conceituação), e tende a diminuir ao longo do mesmo. **Resultado**: Documento de Aquisição de Conhecimento.

Avaliação: Essa atividade visa a avaliar cada um dos passos descritos para garantir a corretitude do desenvolvimento da ontologia (verificação) e garantir que a ontologia representa o domínio de conhecimento definido nas fases de Planejamento e Especificação (validação). **Resultado**: Documento de Avaliação.

Documentação: Todo o processo deve ser documentado com o objetivo de facilitar a manutenção da ontologia e a compreensão da mesma por parte dos usuários finais. **Resultado**: Conjunto de documentos gerados nas outras fases.

Vale ressaltar que as fases de documentação, aquisição de conhecimento e avaliação são realizadas durante todo o processo de desenvolvimento da ontologia.

A Metodologia de Noy e McGuinness

Como dito anteriormente, parte da metolodogia de Noy e McGuinness (2001) foi utilizada para guiar a fase de Conceituação do processo de desenvolvimento da OntoSec. No entanto, a seguir, todos os passos definidos por Noy e McGuinness para o desenvolvimento de uma ontologia são descritos.

Determinar o domínio e o escopo da ontologia: Deve-se decidir a que domínio a ontologia deve atender, com que finalidade, que questões a ontologia deve responder e quem deve usá-la e mantê-la. Todas essas questões devem perdurar durante o processo de desenvolvimento de uma ontologia no sentido de delimitar seu escopo. Técnicas de *brainstorming* são comumente utilizadas. Esse passo se assemelha à fase de Planejamento e Especificação da `Methontology`.

Considerar o reúso de ontologias existentes: Reutilizar ontologias criadas e validadas por outro projetista é bastante válido. Mais relevante ainda se essas ontologias consideram o mesmo domínio da ontologia que se deseja propor. Esse passo se assemelha à fase de Integração da `Methontology`.

Enumerar conceitos importantes da ontologia: É importante gerar uma lista de conceitos sobre os quais a ontologia deve manipular declarações. Essa lista deve ser feita sem se preocupar com sobreposições dos conceitos, com relacionamentos entre conceitos, com atributos que os conceitos podem ter, ou se os conceitos são classes ou não. Esse passo e os outros a seguir se assemelham à fase de Conceituação da `Methontology`.

Definir classes e a hierarquia de classes (ou taxonomia): Esse passo é considerado o mais importante no processo de desenvolvimento de ontologias. Nele os conceitos da ontologia são representados como classes, e os relacionamentos entre as classes são definidos. Nesse passo também é possível definir se entre as classes existem relacionamentos que não representam uma hierarquia, ou seja, relacionamentos que representam eventos entre classes. Por exemplo, na OntoSec um incidente de segurança, que é representado pela classe **SecurityIncident**, pode causar uma ou mais conseqüências, que são representadas pela classe **Consequence**. Nesse caso, o *"causar"* é um relacionamento não-hierárquico. Três abordagens para a definição de hierarquia de classes são sugeridas por Uschold e Gruninger (1996): (i) a abordagem *top-down*, definindo as classes mais gerais e depois as específicas; (ii) abordagem *bottom-up*, que segue o processo inverso; e (iii) a abordagem *middle-out*, que começa por classes internas que vão sendo especializadas e/ou generalizadas. A abordagem utilizada no desenvolvimento da OntoSec foi a *middle-out*.

Definir atributos de cada classe: Isoladamente, classes não fornecem informação suficiente para responder às questões delimitadas na definição do domínio e do escopo da ontologia. Atributos[2] definem uma classe e podem ser adquiridos diretamente da lista de conceitos criada anteriormente. Para cada atributo na lista, é necessário determinar que classe ele descreve e vinculá-lo à mesma. Um atributo deve ser atribuído à classe mais genérica. Todas as subclasses de uma classe herdam seus atributos. Por exemplo, a subclasse **VinhoTinto** herda os atributos da classe **Vinho**.

Definir restrições dos atributos: Atributos podem ter diferentes restrições[3]. Restrições como as que descrevem o tipo de valor armazenado, por exemplo, seqüências de caracteres, inteiros ou booleanos, os valores possíveis que o atributo pode assumir, ou as cardinalidades máxima e mínima. Por exemplo, os valores possíveis do atributo

[2]Um atributo pode ter como sinônimos as palavras *slot*, *role* ou *property*.
[3]Uma restrição pode ter como sinônimos as palavras *facet* ou *role restriction*.

saborvinho na ontologia de vinhos para a classe **Vinho** são *seco*, *demiseco* e *suave*. Além disso, esses valores são mutuamente exclusivos, portanto o atributo ***saborvinho*** tem cardinalidade 1 (exatamente "um").

Definir instâncias: O último passo envolve a criação de instâncias (ou indivíduos) das classes. Definir instâncias envolve escolher uma classe, criar uma instância dessa classe e associar valores aos seus atributos. Por exemplo, um vinho denominado "Lote 43" é uma instância da classe **Vinho** cujo atributo ***saborvinho*** tem valor *seco*. Uma ontologia com um conjunto de instâncias constitui uma base de conhecimento.

No desenvolvimento de ontologias, é importante que a hierarquia de classes esteja correta. No entanto, não existe apenas uma hierarquia correta para um determinado domínio. A hierarquia depende dos possíveis usos que a ontologia terá, do nível de detalhes necessários para uma aplicação que utilizará a ontologia, ou ainda dos requisitos de compatibilidade que a ontologia deve ter com outros modelos do domínio.

Nesse sentido, alguns cuidados devem ser tomados durante a definição da hierarquia de classes da ontologia, como apontam Noy e McGuinness (2001):

- As classes devem ter nomes que representem os conceitos do domínio de conhecimento e não apenas palavras que denotem aqueles conceitos. Se um conceito possui sinônimos, adote o mais comum naquele domínio e acrescente os sinônimos na documentação da ontologia.

- Especializações só devem ser realizadas quando a subclasse criada possuir características (atributos, relacionamentos ou restrições) além daquelas herdadas da superclasse. Além disso, a escolha de criar ou não novas classes também depende do uso que a ontologia terá. Assim, características adicionais nem sempre levam à criação de novas classes. Por exemplo, em uma ontologia de vinhos que será utilizada com uma ontologia de comidas, é possível criar, a partir de uma classe **Vinho**, as subclasses **VinhoTinto**, **VinhoRose** e **VinhoBranco** e as subclasses **VinhoSeco**, **VinhoDemiSeco** e **VinhoSuave**. Essas distinções são naturais no mundo dos vinhos e importantes quando um vinho deve ser combinado com uma determinada comida. Por outro lado, se a ontologia não será utilizada para combinar vinhos com comida, o sabor do vinho pode, por exemplo, ser representado como um atributo da classe **Vinho**, que é herdado pelas subclasses **VinhoTinto**, **VinhoRose** e **VinhoBranco**. Nesse caso, o atributo ***saborvinho*** terá como instâncias *seco*, *demiseco* e *suave*.

- Classes com apenas uma subclasse podem representar um erro de modelagem ou podem representar que a ontologia está incompleta, e classes com muitas subclasses podem requerer a criação de novas classes intermediárias. Afinal, uma ontologia representa o que existe no mundo real. No entanto, se uma especialização não existe no mundo real, então também não deve existir na ontologia.

- Os nomes das classes, atributos, relacionamentos e restrições devem ser padronizados. Ou seja, se o nome de uma classe começa com letra maiúscula e está no singular, todas as classes devem ser representadas assim.

Não existe uma única ontologia correta para um determinado domínio. O desenvolvimento de ontologias é um processo criativo, e as potenciais aplicações da ontologia e a compreensão do domínio que o desenvolvedor possui influenciam nas decisões de projeto e conseqüentemente no modelo da ontologia.

3.2.3 Linguagens de Representação

A princípio, qualquer linguagem pode ser usada para formalizar ontologias. A escolha de uma linguagem deve ser feita com base na sua adequação aos propósitos de representação da ontologia. No entanto, na prática, poucas linguagens têm sido utilizadas para esse fim.

Para o desenvolvimento da OntoSec, duas linguagens foram utilizadas e são descritas a seguir: a linguagem RDF (*Resource Description Framework*), para armazenar os incidentes de segurança, que são as instâncias da OntoSec, e a linguagem OWL (*Web Ontology Language*) (Bechhofer et al., 2004) para formalizar a ontologia. Os incidentes de segurança que instanciam as classes e propriedades da OntoSec foram mantidos separados da ontologia e são dinamicamente integrados quando do processo de validação. O Capítulo 5 explica com mais detalhes esse processo.

Resource Description Framework (RDF)

O uso de metadados (*metadata*) sobre um recurso Web permite o conhecimento de seu significado, características, uso, localização e relacionamentos com outros recursos. O grupo de trabalho em Web Semântica (Berners-Lee et al., 2001) do W3C (*World Wide Web Consortium*) tem dedicado grande parte de seus esforços na descrição e representação de metadados relacionados a qualquer recurso Web. Nesse contexto, o padrão RDF (Beckett, 2004; Klyn e Carroll, 2004) foi definido e envolve três componentes: o modelo de dados, a sintaxe para intercâmbio de metadados e a linguagem de descrição de vocabulários ou esquemas.

RDF é uma aplicação XML que serve como base para o processamento de metadados (dados sobre dados), ou seja, é um *framework* que permite descrever qualquer recurso (e seu conteúdo) na Web. O padrão XML permite estruturar dados em hierarquias modeladas pelo próprio desenvolvedor da aplicação. Documentos escritos em XML podem ser estruturados pelo uso de DTD ou de Esquema XML (*XML Schema*) (Fallside e Walmsley, 2004) (meta-documento para instanciar documentos XML). Tal característica permite que agentes computacionais (inteligentes) interpretem o significado dos dados por meio das *tags* associadas a ele.

Enquanto HTML (*HyperText Markup Language*) descreve conteúdo em conceitos de como os dados são mostrados, por exemplo, a letra "<p>" é uma *tag* que inicia um novo parágrafo, XML descreve conteúdo em conceitos de quais dados são descritos, por exemplo, o conteúdo da *tag"<phonenum>"* é identificado como um número de telefone. Nesse caso, dependendo da aplicação, o número de telefone pode ser armazenado, mostrado ou discado. Portanto, um arquivo XML pode ser processado como um dado por um programa ou pode ser armazenado em uma base de dados, ou, assim como um documento HTML, pode ser simplesmente mostrado. Documentos XML podem ser utilizados em conjunto com documentos HTML em diversas aplicações Web.

RDF, ao contrário de XML, permite a representação de relações entre objetos e possui melhores funcionalidades para representar semântica. Documentos RDF são compostos por declarações sobre os recursos ou objetos representados, sendo que cada declaração é composta pelo **objeto**, uma **propriedade** e o **valor** dessa propriedade, que pode ser outro objeto (relação entre objetos). Assim como XML, RDF também permite que agentes computacionais (inteligentes) interpretem o significado dos dados por meio das *tags* associadas a ele.

A base conceitual de RDF é o seu modelo de dados que, por sua vez, constitui-se de declarações sobre recursos. Uma **declaração** corresponde a uma tripla **sujeito** (*subject*), **predicado** (*predicate*) e **objeto** (*object*). Sujeito é um recurso identificado por uma URI (*Uniform Resource Identifier*)[4]. Um predicado pode ser uma propriedade do recurso ou um relacionamento entre um recurso e um objeto. Um objeto pode ser outro recurso relacionado, ou o valor da propriedade do recurso. Quando valor de uma propriedade, o objeto é um literal, geralmente, uma cadeia de caracteres. O conjunto de triplas forma

[4]URI é um mecanismo padrão de endereçamento e associação de nomes a recursos na Web (Berners-Lee et al., 1998). Essencialmente, uma URI corresponde a uma série de caracteres que identifica e localiza recursos físicos e abstratos de forma unívoca. As URLs (*Uniform Resource Locators*) são casos específicos de URIs nos quais são declarados o protocolo de acesso ao recurso, uma seqüência de caracteres que corresponde ao nome da máquina na qual se encontra o recurso e o próprio recurso em questão.

um grafo RDF, sendo os nós desse grafo o conjunto de sujeitos e objetos, e os arcos o conjunto de propriedades (predicados), como mostra a Figura 3.1.

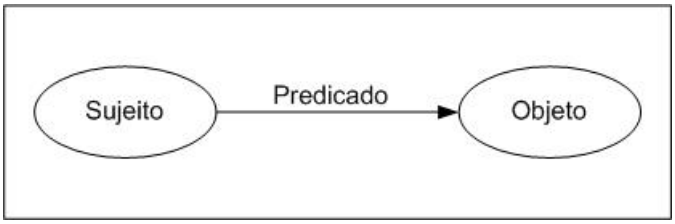

Figura 3.1: Grafo RDF.

O ponto forte do modelo de dados RDF está em sua generalidade: estruturas de dados para hiperdocumentos são facilmente mapeadas para grafos rotulados dirigidos, idéia central desse modelo de dados. Sob a representação de grafo, um recurso é representado por uma elipse identificada por uma URI, o valor da propriedade por retângulo, e a propriedade por um arco que conecta o recurso ao valor da propriedade. Como propriedades também são recursos, elas são rotuladas com uma URI que identifica o espaço de nomes no qual foram definidas.

Enquanto RDF define um modelo para descrição de relações entre objetos em termos de propriedades e valores, o relacionamento entre propriedades e recursos e respectiva semântica é descrito no Esquema RDF (*RDF Schema*) (Brickley e Guha, 2004). O Esquema RDF descreve regras para o uso das propriedades do RDF, definindo um vocabulário de domínio e organizando esse vocabulário em algum tipo de hierarquia. O Esquema RDF permite, ainda, modelar uma aplicação a partir de um esquema conceitual representando hierarquias entre classes e seus relacionamentos. A Figura 3.2 mostra um exemplo de um documento Esquema RDF. O exemplo declara as propriedades entre objetos de um vocabulário do domínio **Animal** (classe). No exemplo, as classes **Mamífero** e **Ave** são subclasses (rdfs:subClassOf) da classe **Animal**.

A partir de XML e Esquema RDF, algumas linguagens para representar ontologia surgiram para a Web Semântica com o principal intuito de permitir a padronização dos dados disponíveis. Gómez-Pérez e Corcho (2002) apresentam um *framework* comparando cada uma dessas linguagens. Para realizar essa comparação, os autores formalizaram em cada uma dessas linguagens, uma ontologia para comércio eletrônico.

Segundo Gómez-Pérez e Corcho, a existência de ferramentas para edição da ontologia com a linguagem selecionada também deve ser considerada na decisão da escolha por uma ou outra linguagem, além da sua adequação aos propósitos de representação da ontologia. Entre essas linguagens estão (vide Figura 3.3):

```
<rdf:RDF
xmlns: rdf="http:llwww.w3.orp,/1999/02/22-rdf-syntax-ns#"
xmlns: rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Description ID="Animal">
<rdf:type
resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<rdfs:subClassOf
resource="http://www.w3.org/TR/WD-rdf-schema#Resource"/>
</rdf:Description>
<rdf:Description ID="Mamífero">
<rdf:type
resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<rdf:subClassOf resource="#Animal"/>
</rdf:Description>
<rdf:Description ID="Ave">
<rdf:type
resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<rdf:subClassOf resource="#Animal"/>
</rdf:Description>
<rdfs:Property rdf:ID="cor">
<rdfs:comment>Cor da penugem </rdfs:comment>
<rdfs:domain rdf:resource="#Ave">
<rdfs:range
rdf:resource="http://www.w3.org/2000/03/example/classes#String">
</rdfs:Property>
<rdfs:Property rdf:ID="habitat">
<rdfs:comment>Local onde é encontrado </rdfs:comment>
<rdfs:domain rdf:resource="#Mamífero">
<rdfs:range
rdf:resource="http://www.w3.org/2000/03/example/classes#String">
</rdfs:Property>
</rdf:RDF>
```

Figura 3.2: Exemplo de um documento Esquema RDF (Moura, 2001).

- *Ontology eXchange Language (XOL)* (Karp et al., 1999).
- *Simple HTML Ontology Extensions (SHOE)* (Luke e Heflin, 2000).
- *Ontology Markup Language (OML)* (Kent, 1999).
- *Ontology Inference Layer (OIL)* (Fensel et al., 2000; Horrocks et al., 2000).
- *DARPA Agent Markup Language + OIL (DAML+OIL)* (Harmelen et al., 2004).
- *Web Ontology Language (OWL)* (Bechhofer et al., 2004).

Web Ontology Language

OWL é uma linguagem de marcação semântica para publicação e compartilhamento de recursos na Web Semântica. Foi desenvolvida pela W3C como um vocabulário estendido

de RDF e é derivada da linguagem DAML+OIL[5]. Como mostra a Figura 3.3, OWL está logo acima da linguagem DAML+OIL.

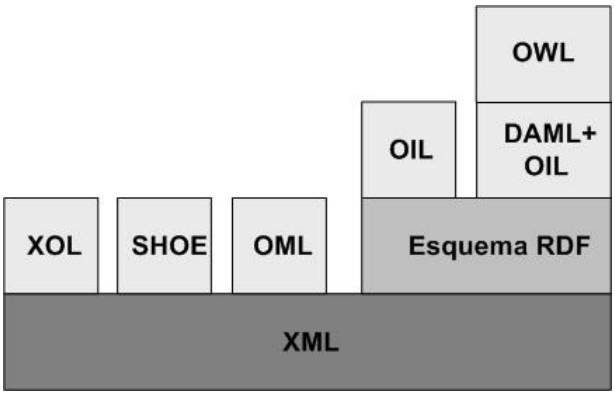

Figura 3.3: Linguagens para descrição de ontologias para WEB Semântica.

O principal objetivo do desenvolvimento da OWL é prover uma linguagem que possa ser utilizada em aplicações que precisam entender o significado das informações e não somente entender como essas informação são apresentadas. A OWL permite uma representação explícita de um vocabulário de conceitos e relacionamentos de um domínio de conhecimento. Dessa maneira, OWL estende o vocabulário de XML, RDF e Esquema RDF, adicionando um vocabulário mais rico para descrever classes, relacionamentos entre classes (por exemplo, disjunção), igualdade ou desigualdade de classes, restrições de cardinalidade e características das propriedades (por exemplo, transitiva, funcional, inversa).

A linguagem OWL segue uma estrutura modular, dividindo-se em três sublinguagens (ou espécies) de acordo com sua capacidade de expressão (Bechhofer et al., 2004; Horridge et al., 2004; McGuinness e Harmelen, 2004).

OWL Lite. É a sublinguagem mais simples. Permite uma definição simples de hierarquia de classes e poucas restrições de propriedades. Por exemplo, *OWL Lite* não permite definir disjunções ou uniões entre classes, e permite apenas cardinalidade 0 ou 1 para as propriedades.

OWL DL (Description Logics). Permite o uso de todos os construtores da OWL, mas com algumas restrições para garantir completude (computabilidade) e decidibilidade (tempo finito) da ontologia. O balanceamento entre expressividade e decidibilidade é garantido pela lógica de descrição (DL), que é um subconjunto

[5]A linguagem DAML+OIL está baseada em Esquema RDF e foi desenvolvida por um comitê conjunto entre Estados Unidos e União Europa (IST) com os principais objetivos de permitir interoperabilidade semântica em XML (DAML) e em recursos Web (OIL).

da Lógica de Primeira Ordem (*First-Order Logic*). A expressividade da lógica de descrição varia de acordo com os construtores OWL utilizados. A notação para representar essa expressividade é apresentada a seguir e indica, por exemplo, que uma ontologia com expressividade `ALH(D)` inclui construtores de atributos lógicos (`AL`), hierarquia de classes (`H`) e tipos de dados (`D`). Como máquinas de inferência apóiam diferentes níveis de expressividade da lógica de descrição, o critério de expressividade deve também ser levado em consideração quando da escolha de uma ou outra máquina de inferência. A expressividade da *OWL DL* é uma variante da lógica de descrição `SHOIN(D)`.

- `AL (Attribute Logic)`: Conjunção, restrições e quantificadores existenciais limitados.
- `C (Complement)`: construtores AL, propriedade de disjunção, e nenhuma restrição quanto aos quantificadores existenciais.
- `R+`: Propriedade de transitividade.
- `S`: Atalho para ALCR+
- `H`: Propriedade de hierarquia.
- `I`: Propriedade Inversa.
- `F`: Propriedade Funcional ou cardinalidade restrita a 1 (um).
- `O`: Lista de indivíduos (instâncias) ou valores.
- `N`: Número maior de restrições.
- `D`: Tipos de dados.

A *OWL DL* permite definir relacionamentos entre instâncias de classes, bem como permite relacionar instâncias de classes a literais RDF ou a tipos de dados do Esquema XML. Com *OWL DL*, por exemplo, classes podem ser construídas por união, intersecção e complemento, ou pela enumeração de instâncias, além de permitir disjunção. No entanto, enquanto uma classe pode ser subclasse de diversas classes (herança múltipla), uma classe não pode ser instância de uma outra classe. Para o desenvolvimento da `OntoSec`, a *OWL DL* foi utilizada.

OWL Full. Permite o uso completo dos construtores da OWL e da sintaxe do RDF, sem restrições. Enquanto *OWL DL* e *OWL Lite* restringem sua sintaxe de classes às classes definidas pelo Esquema RDF e pequenas extensões, por exemplo, owl:equivalentClass, *OWL Full* permite a criação e manipulação de metaclasses via

enumerações, restrições de propriedades e combinações booleanas. No entanto, não há garantia de computabilidade para ontologias desenvolvidas com o uso da *OWL Full*.

As linguagens menos expressivas (*Lite* e *DL*) estão contidas dentro das mais expressivas (*DL* e *Full*), de maneira que uma ontologia definida em uma linguagem menos expressiva é aceita por uma linguagem mais expressiva. No entanto, a recíproca não é verdadeira.

A Sintaxe da OWL

Para definir uma ontologia em OWL, é necessário indicar que vocabulários específicos são utilizados por meio de um conjunto de espaços de nomes XML declarado no início da definição da ontologia (Smith e Welty, 2004). O exemplo a seguir ilustra o trecho inicial de uma ontologia de vinhos[6].

```
1   <rdf:RDF
2       xmlns     ="http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#"
3       xmlns:vin ="http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#"
4       xmlns:food="http://www.w3.org/TR/2003/CR-owl-guide-20030818/food#"
5       xmlns:owl ="http://www.w3.org/2002/07/owl#"
6       xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
7       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
8       xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#">
```

As duas primeiras declarações identificam o espaço de nomes XML associado à ontologia corrente (linhas 2 e 3). A primeira declara que quaisquer nomes sem referência a espaços de nomes XML pertencem à ontologia corrente (linha 2). A segunda declaração associa o espaço de nomes XML da ontologia com o prefixo *vin* para referenciar definições de seu vocabulário (linha 3). A declaração na linha 4 associa o espaço de nomes XML da ontologia de comidas (*food*) com o prefixo *food*. As demais declarações de espaços de nomes XML indicam que elementos com os prefixos *owl*, *rdf*, *rdfs* e *xsd* devem ser entendidos como definições pertencentes aos vocabulários das linguagens OWL, RDF, esquema RDF e esquema XML, respectivamente (linhas 5, 6, 7 e 8).

A maioria dos elementos em uma ontologia OWL trata de classes, propriedades, instâncias de classes e relacionamentos entre instâncias. Os conceitos mais básicos de um domínio de conhecimento devem imaginar classes como raízes de árvores taxonômicas. Tudo declarado em OWL é membro da classe owl:Thing. Assim, cada classe de uma ontologia é subclasse de owl:Thing. Os construtores clássicos para definição e hierarquização

[6]Para ilustrar a sintaxe da linguagem OWL uma ontologia de vinhos é utilizada (`http://www.w3.org/2001/sw/WebOnt/guide-src/wine.owl`.)

de classes são owl:Class e rdfs:subClassOf, respectivamente[7]. Portanto, se X é subclasse de Y, cada instância de X é instância de Y. O exemplo a seguir declara que a classe de vinhos (linha 1) é subclasse da classe de líquidos potáveis (linha 2).

```
1    <owl:Class rdf:ID="Wine">
2      <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
3      ...
4    </owl:Class>
```

A definição de vinhos até este momento indica que vinhos são "líquidos potáveis", portanto sem informação suficiente para criar e interpretar indivíduos dessa classe. Para descrever membros de uma classe, utiliza-se o nome da classe à qual pertence seguido do construtor *rdf:ID* para identificá-lo, como no exemplo a seguir: uvas para o vinho *Merlot* (linha 4) pertencem à classe de uvas de vinho (linha 1) que, por sua vez, são da classe de uvas (linha 2) declarada no espaço de nomes XML de comidas.

```
1    <owl:Class rdf:ID="WineGrape">
2      <rdfs:subClassOf rdf:resource="&food;Grape" />
3    </owl:Class>
4    <WineGrape rdf:ID="Merlot" />
```

O exemplo de ontologia com classes de vinhos e suas instâncias pode ser enriquecido com a associação de propriedades a classes. Propriedades permitem declarar fatos genéricos sobre membros de classes, bem como fatos específicos sobre instâncias. O exemplo a seguir declara que vinhos (linha 2) possuem uma propriedade que indica que estes são feitos de uva (linha 1). Essa propriedade assume valores que pertencem à classe de uvas de vinho (linha 3). O construtor owl:ObjectProperty relaciona instâncias de duas classes, neste caso, instâncias de vinhos e de uvas de vinhos.

```
1    <owl:ObjectProperty rdf:ID="madeFromGrape">
2      <rdfs:domain rdf:resource="#Wine"/>
3      <rdfs:range rdf:resource="#WineGrape"/>
4    </owl:ObjectProperty>
```

Para enriquecer o exemplo anterior, restrições sobre a cardinalidade de propriedades podem ser feitas. O próximo exemplo declara que vinhos (linha 1) são feitos (linha 5) a partir de, no mínimo, um tipo de uva (linha 6). Deve-se ressaltar a utilização de tipos de dados segundo esquemas XML (linha 6) na restrição da propriedade em questão.

[7]Maiores detalhes sobre os construtores fornecidos pelo esquema RDF, consultar (Brickley e Guha, 2004).

```
1   <owl:Class rdf:ID="Wine">
2     <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
3     <rdfs:subClassOf>
4       <owl:Restriction>
5         <owl:onProperty rdf:resource="#madeFromGrape"/>
6         <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
7       </owl:Restriction>
8     </rdfs:subClassOf>
9   </owl:Class>
```

A linguagem OWL fornece construtores para a definição de propriedades transitivas, simétricas, inversas, funcionais e funcionais inversas. Os trechos de ontologias em OWL a seguir ilustram exemplos respectivos. Transitividade indica que se uma região de produção de vinho está localizada em uma região Y, e Y está localizada em uma região Z, então X está localizada na região Z (linhas 1 e 2). Simetria indica que a ordem das premissas não altera o resultado, ou seja, se uma região X produtora de vinho está próxima a uma região Y, então Y está próxima a X (linhas 6 e 7). Inversão indica que existe uma propriedade cuja semântica é oposta à de uma outra propriedade. Neste caso, se uma entidade X produz o vinho Y (linha 14), há uma propriedade inversa que indica que o vinho Y é produzido pela entidade X (linha 16).

Propriedades funcionais são aquelas cujo valor é único. Por exemplo, a propriedade que se refere à entidade produtora de vinho (linhas 11 e 12), já que um vinho não pode ter mais de uma entidade produtora. Propriedades funcionais inversas são aquelas cujo valor se aplica a uma única instância. Neste caso, o vinho Y é produzido por uma única entidade que também possui identificação única (linha 15). Muitas chaves primárias utilizadas em bancos de dados são funcionais e funcionais inversas ao mesmo tempo, como cadastro de pessoa física (CPF), número de passaporte ou número de carteira de motorista.

```
1   <owl:ObjectProperty rdf:ID="locatedIn">
2     <rdf:type rdf:resource="&owl;TransitiveProperty" />
3     <rdfs:domain rdf:resource="&owl;Thing" />
4     <rdfs:range rdf:resource="#Region" />
5   </owl:ObjectProperty>

6   <owl:ObjectProperty rdf:ID="adjacentRegion">
7     <rdf:type rdf:resource="&owl;SymmetricProperty" />
8     <rdfs:domain rdf:resource="#Region" />
9     <rdfs:range rdf:resource="#Region" />
10  </owl:ObjectProperty>

11  <owl:ObjectProperty rdf:ID="hasMaker" />
12    <rdf:type rdf:resource="&owl;FunctionalProperty" />
13  </owl:ObjectProperty>

14  <owl:ObjectProperty rdf:ID="producesWine">
15    <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
```

```
16     <owl:inverseOf rdf:resource="#hasMaker" />
17   </owl:ObjectProperty>
```

Se a simples declaração a seguir for submetida a uma máquina de inferência para a linguagem OWL, é inferido que o elemento declarado (linha 1) é um tipo de líquido potável classificado como vinho, pois possui uma propriedade (linha 2) que é do domínio da classe de vinhos. Além disso, esse vinho chamado *LindemansBin65Chardonnay* tem propriedades que indicam sua entidade produtora, bem como a região na qual ela se encontra e quais são as regiões adjacentes.

```
1    <owl:Thing rdf:ID="LindemansBin65Chardonnay">
2      <madeFromGrape rdf:resource="#ChardonnayGrape" />
3    </owl:Thing>
```

3.3 Ambientes para Modelar e Avaliar Ontologias

Desenvolver uma ontologia, como já mencionado, não é uma tarefa fácil. Envolve diversos passos e gera vários documentos. Para facilitar esse desenvolvimento, diversos ambientes são encontrados na literatura para auxiliar no processo de desenvolvimento de ontologias. Esses ambientes visam a apoiar todos os passos realizados pelo desenvolvedor, desde a modelagem até a avaliação da ontologia.

Dentre os ambientes utilizados para modelar ontologias, têm-se: Protégé[8], OntoSaurus[9], CommonKADS[10], WebOnto[11], OntoEdit[12], Ontolingua-Server[13] e WebODE[14]. Todos esses ambientes suportam diversas linguagens para representar ontologias, como XML, DAML+OIL, OIL, OWL, RDF-Schema e Ontolingua. Além desse suporte às linguagens de representação, muitas dessas ferramentas também permitem que diferentes máquinas de inferência possam ser integradas e utilizadas para avaliar a ontologia, como Pellet (Sirin et al., 2006), RacerPro[15], FaCT++[16], e diferentes linguagens de consultas para ontologias, como RDQL (*RDF Data Query Language*) (Seaborne, 2004) e SPARQL (*SPARQL Query Language for RDF*)(Prud'hommeaux e Seaborne, 2006). Essas linguagens permitem que o desenvolvedor faça perguntas para a ontologia, auxiliando-o

[8] http://protege.stanford.edu/.
[9] http://www.isi.edu/isd/ontosaurus.html.
[10] http://hcs.science.uva.nl/projects/kads22/.
[11] http://kmi.open.ac.uk/projects/webonto/.
[12] http://www.ontoknowledge.org/tools/ontoedit.shtml.
[13] http://www.ksl.stanford.edu/software/ontolingua/.
[14] http://kw.dia.fi.upm.es/wpbs/.
[15] http://www.sts.tu-harburg.de/\~r.f.moeller/racer/.
[16] http://owl.man.ac.uk/factplusplus/.

no processo de avaliação. Essas perguntas são aquelas "**questões de competências**" que podem ser definidas durante o processo de desenvolvimento da ontologia.

A seguir, nesta seção, a ferramenta Protégé, a máquina de inferência Pellet, e a linguagem de consulta SPARQL são descritas com mais detalhes, pois foram utilizadas durante o desenvolvimento e avaliação da OntoSec.

3.3.1 A Ferramenta Protégé

A ferramenta Protégé tem sido desenvolvida pela Universidade de Stanford (*Stanford Medical Informatics*) desde 1997 com o principal objetivo de auxiliar desenvolvedores e especialistas na modelagem de aplicações baseadas em conhecimento utilizando ontologias (Gennari et al., 2003). Uma grande comunidade de desenvolvedores, sejam eles da área acadêmica ou da indústria ou de governos, utilizam a Protégé para modelar ontologias nas mais diversas áreas do conhecimento.

Como núcleo de sua implementação, a ferramenta possui um conjunto de estruturas que apóiam a criação, visualização e manipulação de ontologias em diferentes formatos. Ela é desenvolvida utilizando a linguagem Java com código aberto, e provê um ambiente *plug-and-play* que a torna flexível para o desenvolvimento de aplicações, possibilitando que *plugins* e APIs (*Application Programming Interface*) Java possam ser incorporadas. Esses *plugins* e APIs podem ser utilizados para modificar ou estender o comportamento da ferramenta. As APIs podem ainda ser diretamente utilizadas por aplicações externas para acessar bases de conhecimento sem a Protégé.

A Protégé também possibilita que código Java seja gerado a partir das classes e relacionamentos da ontologia, facilitando o desenvolvimento de aplicações com base na ontologia. Além disso, é possível utilizar de maneira integrada à Protégé diferentes máquinas de inferências, como Pellet e RacerPro, para avaliar uma ontologia, diferentes linguagens, como RDQL e SPARQL, e diferentes ferramentas de visualização da hierarquia de classes e relacionamentos da ontologia, como Ontoviz[17] e OWLViz[18].

A Protégé apóia dois tipos de modelagem de ontologias:

- **Editor *Protégé-Frames***: Possibilita que usuários desenvolvam ontologias baseadas em *frames* de acordo com o protocolo OKBC (*Open Knowledge Base Connectivity*). Nesse modelo, a ontologia consiste de um conjunto de classes organizadas em uma hierarquia que representa os conceitos de um domínio, um conjunto de

[17] `http://protege.stanford.edu/plugins/ontoviz/ontoviz.html`.
[18] `http://www.co-ode.org/downloads/owlviz/`.

slots (ou atributos) associados a uma classe para descrever suas propriedades, e um conjunto de instâncias dessas classes.

- **Editor *Protégé-OWL***: Possibilita que usuários desenvolvam ontologias para a Web Semântica utilizando a linguagem OWL. Nesse caso, todas as propriedades e construtores da OWL podem ser utilizados durante a modelagem. Outras linguagens, como RDF e XML, também podem ser utilizadas. Esse editor gera o código OWL da ontologia automaticamente à medida que a ontologia está sendo modelada. O editor Protégé-OWL foi utilizado no desenvolvimento da OntoSec.

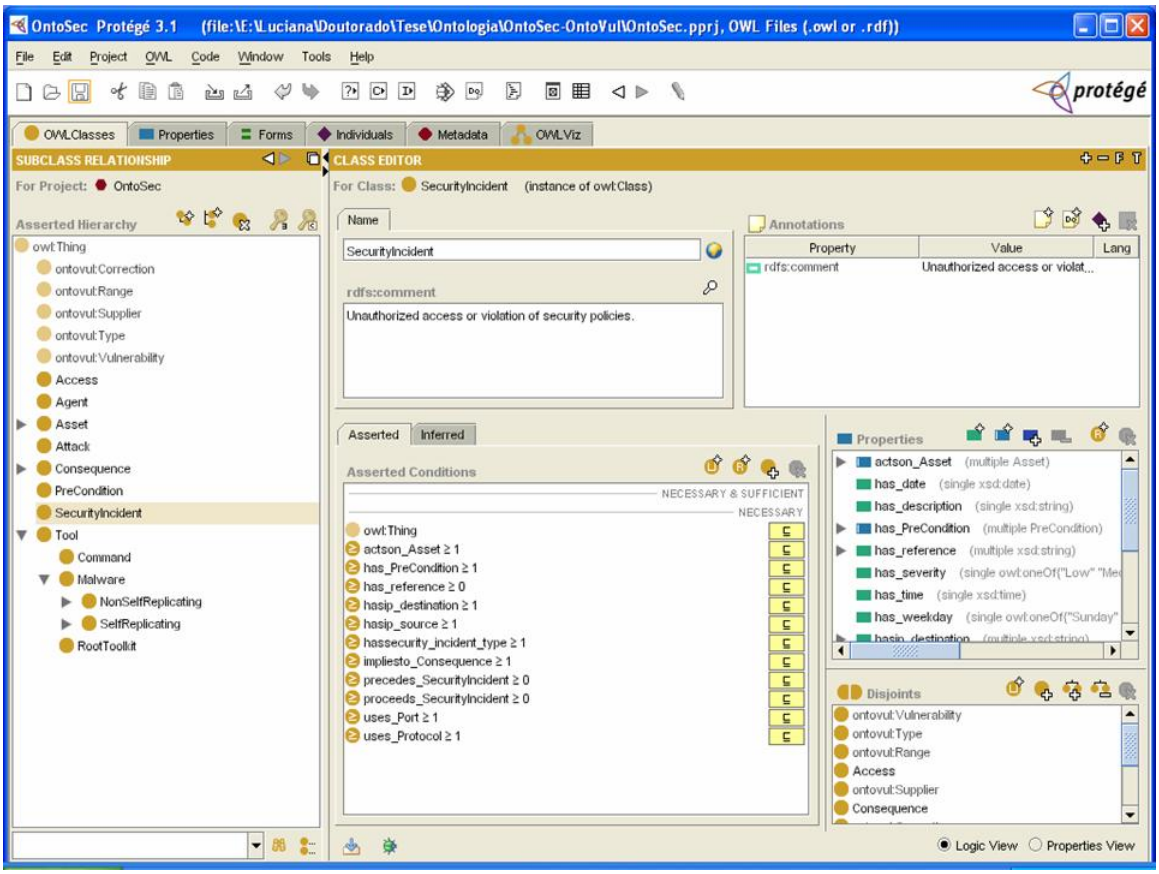

Figura 3.4: *Screenshot* da ferramenta Protégé versão 3.1.

A Figura 3.4 apresenta um *screenshot* da Protégé-OWL com as classes, atributos e suas restrições, e os relacionamentos da OntoSec que aparecem na aba ***OWL Classes***. A ferramenta Protégé mostra do lado esquerdo a hierarquia de classes da ontologia, denominada *Asserted Hierarchy*. A classe **owl:Thing** é criada por *default* como raiz da hierarquia e todas as outras classes são criadas abaixo dela. Do lado direito, no *Class Editor* é possível criar os atributos e relacionamentos (*Properties*) e as restrições (*Asserted*

Conditions) de uma classe. No campo (*Properties*), os atributos são identificados com a cor verde, enquanto que os relacionamentos são indicados com a cor azul. Nesse mesmo campo, é possível estabelecer as restrições dos atributos e dos relacionamentos.

Ainda no *Class Editor*, no canto direito abaixo, é possível definir que uma classe é distinta (*disjoint*) de outra. Isso significa que instâncias de uma determinada classe não podem fazer parte de outra classe.

As outras abas da ferramenta apresentam as seguintes informações[19]:

- ***Properties***: Permite a criação das propriedades da ontologia.
- ***Forms***: Mostra as propriedades criadas para as classes da ontologia.
- ***Individuals***: Permite criar instâncias para as classes da ontologia.
- ***Metadata***: Mostra todos os espaços de nomes da ontologia e permite importar outras ontologias.
- ***OWLViz***: Essa aba mostra um *plugin* que permite visualizar graficamente a hierarquia de classes.

3.3.2 A Máquina de Inferência Pellet

As máquinas de inferência normalmente oferecem diferentes serviços de consultas que podem ser utilizados para questionar uma ontologia. Esses serviços podem questionar classes, relacionamentos, atributos ou instâncias, realizando conclusões a respeito do domínio. Por exemplo, dada uma classe C, (i) determinar se um indivíduo I é instância de C, (ii) determinar todos os indivíduos de C, (iii) determinar se uma classe D é subclasse de C, ou (iv) se uma classe E é subclasse de D e D é subclasse de C, então E também é subclasse de C (transitividade) (Tempich e Volz, 2003).

Pellet é uma máquina com código aberto escrita em Java utilizada para avaliar ontologias desenvolvidas utilizando a linguagem *OWL DL*. Como *OWL DL* está baseada na lógica de descrição, Pellet utiliza os jargões da lógica de descrição para analisar a expressividade e as características de uma ontologia (Sirin et al., 2006; Vieira et al., 2005), que são:

- ***ABox (Assertional Box)***: Representa as assertivas (*assertions*) sobre os indivíduos da ontologia, tais como tipo e valor. *ABox* contém o conhecimento que

[19] Os *plugins* que são integradas à Protégé aparecem como abas na ferramenta.

especifica os indivíduos do domínio, sendo, assim, a instanciação da estrutura de conceitos. Existem dois tipos de declarações:

  - Declaração de Conceitos: C(a). Declara que "a" é um indivíduo do conceito C. Por exemplo, Vinho(MarcusJames).
  - Declaração de Papel: R(a,b). Declara que o indivíduo "a" está relacionado com o indivíduo "b" por meio da propriedade "R". Por exemplo, produz(VinicolaAurora,MarcusJames).

- ***TBox (Terminological Box)***: Representa os axiomas sobre as classes (conceitos) da ontologia, tais como subclasses, disjunção, equivalência e transitividade. *TBox* representa as características gerais dos conceitos de um domínio, que são grupos de indivíduos semelhantes.

- ***KB (Knowledge Base)***: Representa a base de conhecimento (ou a ontologia completa), que é a combinação entre *ABox* e *TBox*.

A partir de uma máquina de inferência como a Pellet, é possível avaliar se uma ontologia está de acordo com serviços de inferência definidos pela lógica de descrição, tais como:

- **Consistência**: Visa a garantir que a ontologia não contém fatos contraditórios, avaliando a consistência de uma *ABox* com relação a um *TBox*.

- **Classificação**: Visa determinar a hierarquia de conceitos (classes) da ontologia.

- **Realização** (*Realization*): Visa a encontrar na base de conhecimento as instâncias de um indivíduo.

Pellet é uma máquina de inferência que possui uma implementação completa para a linguagem *OWL/DL*, incluindo análise das instâncias da ontologia, dos tipos de dados e das restrições. Por isso, ela foi utilizada para avaliar a OntoSec.

3.3.3 A Linguagem SPARQL

RDF tem sido largamente utilizado para representar recursos na Web. Além de permitir a representação de recursos, o padrão RDF também possibilita a integração entre diferentes fontes de dados. Uma vez que esses recursos estão armazenados de forma padrão, desenvolvedores e usuários podem adquirir informação e conhecimento a respeito

dos recursos armazenados utilizando linguagens de consulta específicas para recursos em RDF. Dentre essas linguagens, tem-se a SPARQL, que apesar de ainda estar em processo de padronização pelo W3C, já é uma linguagem bastante utilizada para consultas em aplicações da Web Semântica.

Com a linguagem SPARQL é possível realizar consultas em dados em RDF armazenados tanto em arquivos ".rdf" quanto em bancos de dados. A linguagem tem a sintaxe básica semelhante à linguagem SQL (*Structured Query Language*), ou seja, utiliza a estrutura *SELECT* **X** *FROM* **Y** *WHERE* **Z**. As consultas são realizadas a partir do grafo RDF, que é composto pela tripla sujeito (recurso), predicado (propriedade) e objeto (recurso ou literal). SPARQL também possui comandos como *ORDER BY*, *UNION*, *DISTINCT*, além de agrupamentos, restrição de valores e extração de informação a partir de mais de uma base RDF.

SPARQL provê as seguintes facilidades: (i) extrair informação na forma de URIs, *blank nodes*, que são aqueles nós de um grafo RDF que não possuem URI, e *plain* e *typed literals*, que são, respectivamente, aqueles que não estão associados ao tipo de dados que armazenam e aqueles que são associados ao tipo de dados que armazenam; (ii) extrair subgrafos RDF; e (iii) construir novos grafos RDF a partir das consultas realizadas.

O exemplo abaixo mostra uma consulta SPARQL simples para encontrar o título de um livro. A cláusula *SELECT* identifica a variável que irá aparecer no resultado da consulta, e a cláusula *WHERE* apresenta a tripla RDF que deverá ser satisfeita.

```
1 SELECT ?title
2  WHERE {<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .}
```

Para a consulta, a instância está representada como mostra o exemplo a seguir. Assim, o resultado é uma tabela com o ***title*** "SPARQL Tutorial".

```
1 <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

A mesma consulta pode ser escrita de outra forma. Como mostra a linha 1, antes das cláusulas *SELECT* e *WHERE*, é possível colocar um "prefixo" que identifica o espaço de nomes utilizado. No caso, o espaço de nomes é do Dublin Core (DC)[20].

```
1  PREFIX  dc: <http://purl.org/dc/elements/1.1/>
2  SELECT  ?title
3  WHERE   { <http://example.org/book/book1> dc:title ?title }
```

[20]O Dublin Core é uma iniciativa de desenvolver interoperabilidade de padrões de metadados e desenvolver vocabulários de metadados para descrever recursos facilitando a descoberta de conhecimento. Mais informações em `http://dublincore.org`.

Para validar a OntoSec, uma aplicação de consultas a incidentes de segurança armazenados em RDF foi desenvolvida. Essa aplicação armazena os incidentes em um banco de dados MySQL e utiliza a linguagem SPARQL para realizar as consultas. Para armazenar os dados dos incidentes no banco de dados, o *framework* Jena[21] foi utilizado. O Jena é um *framework* Java largamente utilizado para desenvolvimento de aplicações da Web Semântica, e provê suporte para os padrões RDF e RDFS, e para as linguagens OWL e SPARQL.

3.4 Trabalhos Relacionados

No domínio de segurança, é possível encontrar algumas ontologias, tais como:

- A *Core Security Ontology* baseada em políticas de segurança definida por Schumacher (2003). Essa ontologia apresenta um modelo genérico de segurança que pode ser utilizado para especificar padrões (*patterns*) de segurança.
- A ontologia para informações de segurança desenvolvida por Raskin et al. (2001). Os pontos principais da ontologia estão relacionados ao formalismo para definir os conceitos envolvidos no domínio de segurança. Assim como Schumacher, Raskin *et al.* têm uma visão de alto nível e abstrata dos problemas de segurança.
- A ontologia *Target-Centric* para representar alertas de ataques registrados por sistemas de detecção de intrusão desenvolvida por Undercoffer e Pinkston (2002)
- A ontologia definida por Filman e Linden (1996) para representar a interoperabilidade entre agentes de software, chamados *safebots*. Filman e Linden definiram uma linguagem chamada *OntoSec* (*Ontology for Security*) para especificar esses agentes e descrever as informações enviadas entre eles.
- A ontologia para representar alertas de vulnerabilidades desenvolvida por Brandão (2004).

A seguir, esses trabalhos são descritos.

3.4.1 A Ontologia de Schumacher

A principal preocupação de Schumacher no desenvolvimento da *Core Security Ontology* foi estabelecer um mapeamento conceitual de políticas de segurança com conceitos utilizados

[21] `http://jena.sourceforge.net/`.

no dia a dia do gerenciamento de segurança. A ontologia representa os conceitos chave de segurança (denominados por Schumacher como *top-level security concepts*) e como eles se relacionam. Assim, dependendo da organização e de sua política de segurança, esses conceitos podem ser especializados e novos relacionamentos podem ser criados, ou seja, a ontologia é facilmente estendida.

Para definir quais conceitos seriam representados, glossários, artigos e livros da área de segurança foram estudados. A principal preocupação nessa fase foi adotar a nomenclatura padrão utilizada na área de segurança para desenvolver a ontologia. Além dessa pesquisa, profissionais de segurança também foram entrevistados (utilizando questões de competência) para auxiliar na definição dos conceitos da ontologia. Todos os conceitos possuem uma definição em língua natural que também está baseada na literatura de segurança. Para representar a *Core Security*, Schumacher não utilizou nenhuma linguagem de representação comumente utilizada para formalizar ontologias.

Os principais conceitos da *Core Security* estão descritos a seguir e a Figura 3.5 mostra como eles se relacionam.

Asset: Representa os recursos (ou ativos) que têm valor para uma organização e podem sofrer ameaças e ataques, como uma aplicação, um equipamento, uma rede ou um funcionário.

Stakeholder: Representa a organização ou pessoa que atribui valor a um ativo e também define as necessidades de segurança.

Security Objective: Representa as necessidades de segurança que uma organização define para um ativo, como graus de confidencialidade, de integridade e de disponibilidade.

Threat: Representa potenciais quebras de segurança de um ativo. Uma ameaça só pode ser considerada um problema quando as necessidades de segurança estão bem definidas. Ameaças podem ser incêndios ou erros humanos.

Attack: Caracteriza as ações que violam a segurança de um ativo explorando vulnerabilidades.

Attacker: Caracteriza a "entidade" que realizada ataques.

Vulnerability: Caracteriza alguma fraqueza que pode ser explorada por um atacante para quebrar a segurança de um ativo.

Countermeasure: Representa as ações realizadas por uma organização a fim de proteger os ativos contra ameaças e ataques, como por exemplo, utilizar apenas programas legalizados, manter sistemas sempre atualizados para evitar eventuais vulnerabilidades ou restringir acessos a determinados locais da organização.

Risk: Representa a probabilidade de um ataque ter sucesso. A estimativa de risco é utilizada a fim de estimar os esforços despendidos para proteger os ativos (custo, tempo, etc.). Para determinar se um risco de segurança é aceitável, é preciso determinar as conseqüências da perda de um ativo e a probabilidade da ocorrência de um ataque.

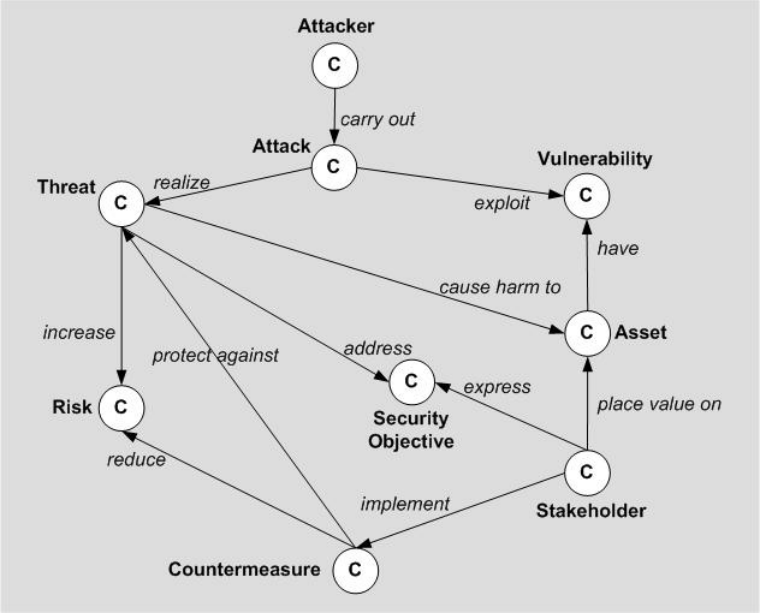

Figura 3.5: Conceitos e relacionamentos da *Core Security Ontology*.

A ontologia definida por Schumacher apresenta conceitos de mais alto nível relacionados à segurança e definidos a partir de uma política de segurança. Como uma política de segurança depende das metas de segurança que uma organização possui, as possíveis extensões da ontologia podem ser bastante diversas, principalmente com relação aos ativos e seus riscos, uma vez que um ativo representa o bem mais importante da organização.

Como contribuição para a *Core Security* seria interessante que as ferramentas que um atacante pode utilizar para realizar ataques também tivessem um conceito de alto nível para representá-las, como por exemplo, **Tool**, que poderia ser especializada dependendo de cada organização.

3.4.2 A Ontologia de Raskin et al.

Com o principal objetivo de possibilitar que informações de segurança armazenadas em língua natural pudessem ser utilizadas para analisar ataques e incidentes de segurança, Raskin *et al.* definiram uma ontologia para representar os conceitos do domínio de segurança e seus significados. Assim, seria possível, por exemplo, que um processador de língua natural baseado em uma ontologia de segurança pudesse "ler" (*text mining*) informações de *logs* gerados por diversas ferramentas de segurança e saber se um ataque está ocorrendo, possibilitando que ações de defesa ou prevenção de ataques sejam executadas de maneira mais eficiente. Ainda, seria possível que esse mesmo processador "lesse" alertas de vulnerabilidades ou relatórios de incidentes para saber quais as vulnerabilidades mais exploradas ou quais os incidentes mais comuns.

A ontologia define de maneira única e formal o significado dos conceitos relacionados ao domínio de segurança, ou como os autores dizem, define uma linguagem comum para representar o significado dos conceitos. Essa linguagem comum definida por Raskin *et al.* possui dois componentes: (1) conjunto de conceitos relacionados à segurança, e (2) método de classificação desses conceitos, ou seja, uma taxonomia. A partir dessa linguagem, é possível organizar e sistematizar ataques.

Aproximadamente, 400 conceitos estão representados, dos quais alguns são citados a seguir: *attack*, *availability*, *CERT*, *buffer overflow*, *firewall*, *cookie*, *hardware*, *spoof*, *virus* e *vulnerability*. É interessante notar que os conceitos vão desde os tipos de ataques como *spoof* até institutos de pesquisa em segurança como o *CERT*. Esses conceitos foram definidos a partir da literatura e de glossários da área de segurança. Todos os conceitos têm representação léxica na língua inglesa utilizando a Forma Normal de Backus (BNF - *Backus-Naur Form*).

A Figura 3.6 mostra uma entrada ontológica de um dos conceitos de segurança, no caso **"Computer Security"** (Raskin et al., 2001). A figura apresenta o significado do conceito, sua descendência direta (*IS-A*), e as propriedades herdadas de outros conceitos. Essas propriedades são identificadas por *SEM*. Por exemplo, o conceito **"Computer Security"** não é composto por outros conceitos (*HAS-PARTS*) ou compõe algum conceito (*PART-OF*).

Além dos objetivos principais de compartilhar conhecimento comum de informações de segurança de forma única, possibilitar avaliações de ataques e determinar seus efeitos e ações necessários para recuperação, Raskin *et al.* estão preocupados com formalismos, terminologia e conceituação de segurança como um todo.

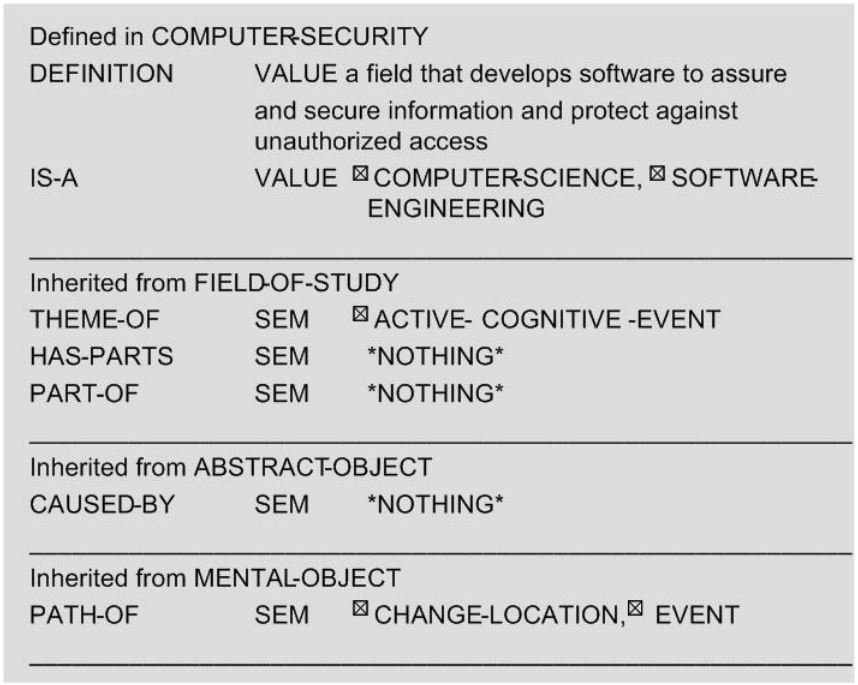

Figura 3.6: Entrada ontológica para o conceito **Computer Security**.

3.4.3 A Ontologia Target-Centric

Utilizando a linguagem DAML+OIL, Undercoffer e Pinkston definiram uma ontologia para representar de maneira única os alertas de segurança relatados por sistemas de detecção de intrusão. A ontologia, denominada *Target-Centric*, tem como principais objetivos facilitar a interoperabilidade entre IDSs, e possibilitar que esses sistemas possam "raciocinar" a partir dos alertas relatados, detectando e minimizando intrusões.

Para definir os conceitos e relacionamentos da ontologia, uma análise empírica foi realizada em mais de 4.000 classes de ataques[22]. A partir dessa análise, os principais atributos, características e relacionamentos dos ataques foram utilizados para o desenvolvimento da ontologia. A Figura 3.7 mostra algumas das classes e os relacionamentos da ontologia *Target-Centric*.

Como mostra a Figura 3.7, a ontologia está baseada no alvo de um ataque, que é representado pela classe **Host**. A classe **Host** se relaciona com as classes **State** e **Intrusion** por meio dos relacionamentos *current state* e *victim of*, respectivamente. A classe **State** descreve o estado de um *host* durante um ataque. Esse estado é representado pelas classes **Network**, **System** e **Process**. Por exemplo, a classe **Network** possui como

[22]Boletins do CERT/CC e do NVD sobre as vulnerabilidades exploradas por determinados tipos de ataques foram utilizados.

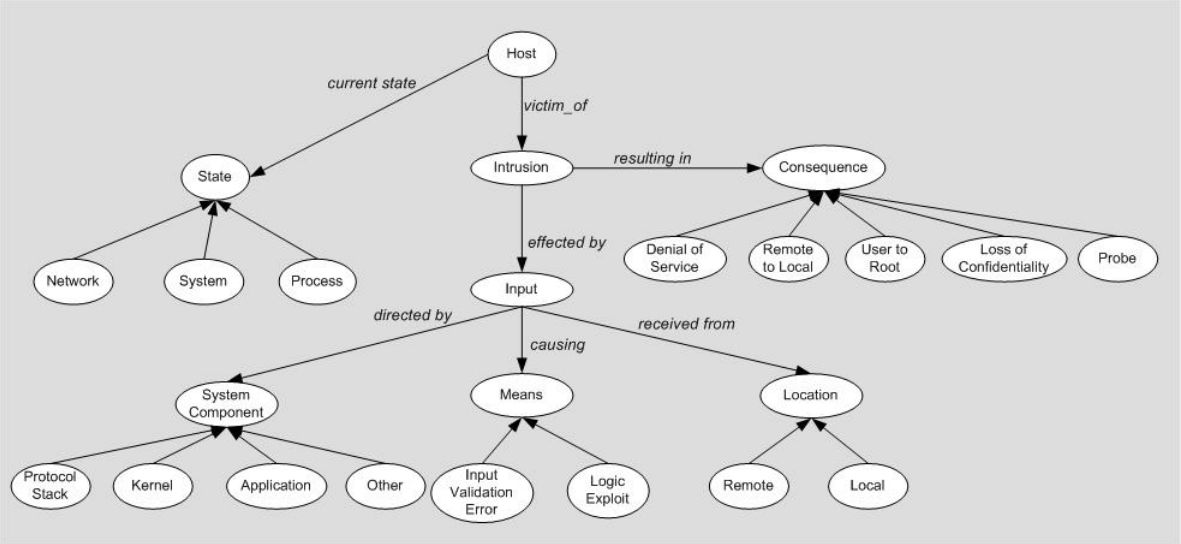

Figura 3.7: Alguns conceitos e relacionamentos da ontologia *Target-Centric* (Undercoffer et al., 2004).

atributo o número máximo de conexões TCP que o *host* pode abrir, e a classe **System** possui atributos que mostram o uso de memória e de processador. A classe **Intrusion** representa os ataques que um *host* pode sofrer e se relaciona com as classes **Input** e **Consequence** por meio dos relacionamentos *effected by* e *resulting in*, respectivamente. A classe **Consequence** representa os tipos mais comuns de conseqüências que um ataque pode ter, enquanto que a classe **Input** possui relacionamentos com as classes **System Component** (*directed to*), **Means** (*causing*) e **Location** (*received from*). A classe **System Component** representa os componentes mais freqüentemente atacados em um sistema, enquanto que as classes **Means** e **Location** representam, respectivamente, os meios mais utilizados para execução de um ataque (**Input Validation Error** e **Logic Exploit**) e de onde o ataque é executado, local ou remotamente.

A ontologia *Target-centric* representa informações de ataques relacionados ao tráfego TCP/IP, incluindo apenas pacotes TCP ou UDP. Assim, a ontologia foi definida com base nas observações e análises dos tipos de ataque mais comuns em um sistema. No entanto, mesmo sendo uma ontologia específica para o tráfego TCP/IP identificado por IDSs, ela pode ser estendida para representar novos meios de ataque, novas conseqüências ou novos componentes. Além disso, a ontologia permite que exista uma coalizão entre diferentes IDSs (interoperabilidade), uma vez que esses IDSs podem compartilhar um vocabulário comum sobre os ataques que um sistema pode sofrer (Undercoffer et al., 2003).

A ontologia tem um papel importante para a definição de um vocabulário comum para representar alertas de intrusão utilizando uma linguagem padrão do W3C para a Web

Semântica. No entanto, se IDSs estiverem integrados a outras ferramentas que geram alertas de segurança, como um *firewall*, por exemplo, o problema de interoperabilidade de alertas ainda persiste em um nível operacional. Nesse caso, um sistema de gerenciamento de segurança que pudesse ler alertas de vários formatos e mapeá-los para um formato único (uma ontologia) possibilitaria que os administradores de segurança pudessem tomar decisões e fazer correlações de maneira mais eficiente em um nível gerencial.

3.4.4 A Ontologia para Agentes de Software

Para Filman e Linden, os controles de segurança em software devem ser realizados de forma ativa, permitindo que agentes (ou *safebots - software robots*) possam monitorar os componentes de um sistema, possam se comunicar, e possam de maneira inteligente planejar ou adaptar ações para que metas de segurança sejam alcançadas. Esses agentes podem ser programados para realizar autenticações, controles de acesso, detecção de intrusão, buscar por vulnerabilidades no sistema, entre outros controles.

Com o intuito de implementar essa idéia de agentes, Filman e Linden (1996) definiram uma linguagem chamada *OntoSec* (*Ontology for Security*) que permite definir como os dados utilizados na comunicação entre agentes são representados, permite especificar os agentes, e possibilita que requisitos de segurança, ações e o conhecimento dos agentes sejam representados. A linguagem possibilita uma representação formal da especificação dos agentes e como eles se comunicam por meio de primitivas que representam os conceitos do domínio de segurança, utilizando lógica de predicados. A *OntoSec* descreve, por exemplo, ps protocolos, as propriedades de segurança de recursos e os componentes de um sistema, os privilégios de usuários, entre outros. A Figura 3.8 mostra uma declaração da *OntoSec*.

A *OntoSec* está integrada a dois outros componentes, *Swathe* e *SecLib*, formando um sistema definido como *SafeBots*. Esse sistema permite que os agentes sejam implementados pelo módulo *Swathe* com base nas especificações definidas pela *OntoSec*, e com base nos fragmentos de códigos de agentes (componentes reusáveis) armazenados pelo módulo *SecLib*. O módulo *Swathe* também utiliza a organização física dos sistemas (onde as aplicações estão rodando) e as definições de interface entre os agentes para a implementação.

O uso de uma ontologia de especificar agentes permite formalizar a maneira como esses agentes são desenvolvidos e como eles se comunicam. Além disso, o uso de agentes distribuídos e móveis para coletar informações de segurança em um sistema é bastante interessante. No entanto, é importante considerar que se o número de agentes for elevado,

```
John ∈ TrustedFriends (LockheedMartin)
May(p,Write,DB42,x,y) → May(p,Read,DB42,x,y)
May(p,Read,item(DB42,Salary(q)) → (p=q | WorksFor(p,q))
May(p,s,OnMachine(K,Shell),x,y) &
UserProgram(m) → May(p,x,OnMachine(K,m),x,y)
Owner(p,r) → Vm.May(p,m,r,x,y)
FailedPasswordTries(s,h,r) > 5 →
    Notify(SessionHolders(s),PasswordHacking(s)) &
    ∀r,x,y. ~May(User(s),s,r,x,y)
Goal(Suspicious(p) → ~May(p,r,a,x,y))
```

Figura 3.8: Exemplo de uma declaração utilizando a linguagem *OntoSec*.

o desempenho do sistema pode ser comprometido e o trabalho do administrador de segurança fica mais complexo. Uma vez comprometido, o sistema pode ficar mais vulnerável a ataques. Outro ponto que deve ser considerado é que o próprio agente pode ser alvo de um ataque. Assim, é importante que esses agentes sejam inteligentes para pode identificar quando eles mesmos estão sofrendo um ataque.

3.4.5 A Ontologia de Vulnerabilidades

Diferentes institutos de pesquisas têm realizado esforços no sentido de catalogar e classificar dados e informações relacionados à segurança computacional. O projeto CVE, por exemplo, apresenta um padrão para nomenclatura de vulnerabilidades que facilita a identificação de uma mesma vulnerabilidade em diferentes ferramentas (Mann e Christey, 1999). No entanto, o CVE não agrega semântica às informações relacionadas aos dados armazenados, dificultando, assim, uma correlação.

Nesse contexto, Brandão (2004) desenvolveu uma ontologia para representar alertas de vulnerabilidades com o principal objetivo de auxiliar a interoperabilidade semântica entre diferentes ferramentas de segurança. Como ponto de partida para o desenvolvimento da ontologia de vulnerabilidades, Brandão utilizou o catálogo de vulnerabilidades mantido pelo CVE, por ser um padrão de *facto*, aberto e bastante aceito pela comunidade. Além do CVE, a base de dados da NVD foi utilizada no processo de pesquisa por características das vulnerabilidades que seriam representadas na ontologia.

Para o desenvolvimento da Ontologia de Vulnerabilidades, Brandão utilizou como guia o modelo proposto por Noy e McGuinness (Noy e McGuinness, 2001) apresentado na Seção

3.2.2. Para modelar a ontologia, a ferramenta Protégé 2.0 e a linguagem *OWL DL* foram utilizadas.

Para validar a ontologia, entradas da base CVE foram cadastradas. O cadastro foi feito na própria ontologia utilizando a ferramenta Protégé. Foram cadastradas 117 vulnerabilidades indexadas pelo NVD durante o mês de março de 2003. A validação consistiu da elaboração de questões feitas à ontologia a fim de descobrir informações relevantes das vulnerabilidades cadastradas. As questões foram realizadas no ambiente de execução da Protégé, utilizando a interface para consultas em RDQL.

A Figura 3.9 mostra as principais classes e relacionamentos da Ontologia de Vulnerabilidades.

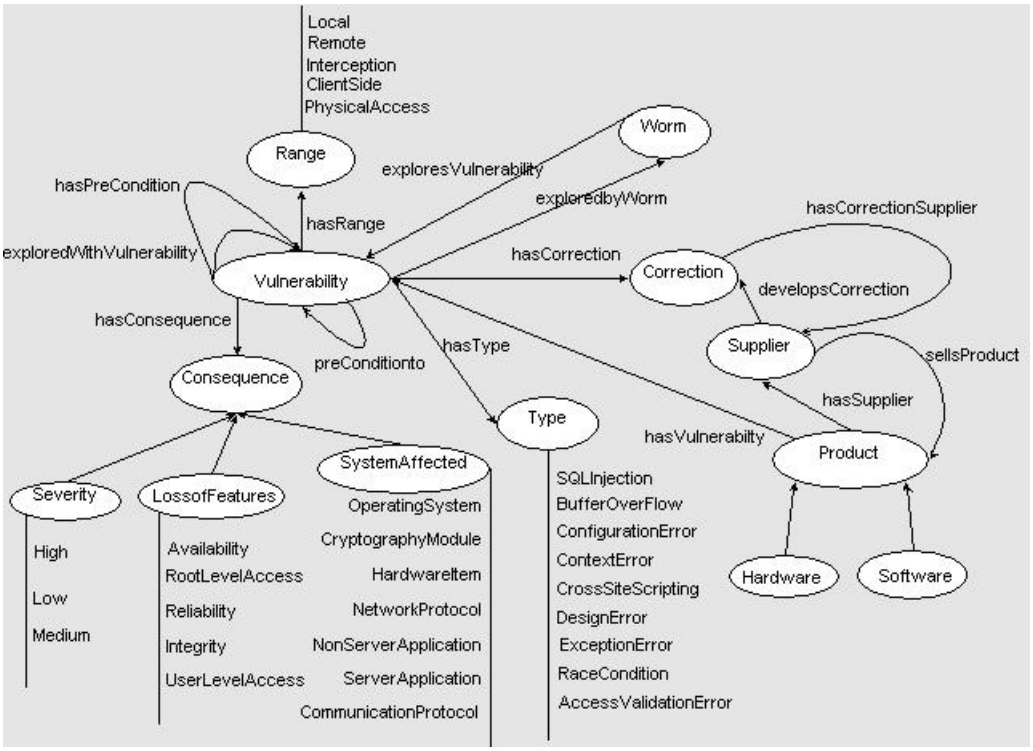

Figura 3.9: Conceitos e relacionamentos da Ontologia de Vulnerabilidades.

Como mostra a Figura 3.9, os principais conceitos (classes) e relacionamentos da Ontologia de Vulnerabilidades são:

Vulnerability: Representa as vulnerabilidades. A classe possui atributos que armazenam o nome da vulnerabilidade (código CVE), breve descrição, data de divulgação e referências sobre a vulnerabilidade que podem ser, por exemplo, *links* para páginas com descrições mais detalhadas. A partir dela partem os relacionamentos que informam qual o tipo da vulnerabilidade (*hasType*), sua abrangência (*hasRange*),

suas conseqüências (*hasConsequence*), suas correções (*hasCorrection*), quais *worms* podem explorá-las (*exploredbyWorm*) e com quais outras vulnerabilidades uma vulnerabilidade pode se relacionar (auto-relacionamento). Os relacionamentos *hasPreCondition* e *preConditionto* indicam que uma (ou mais) vulnerabilidade(s) deve(m) existir para que outras sejam exploradas (relacionamentos inversos), e o relacionamento *exploredWithVulnerabilities* indica que vulnerabilidades podem ser exploradas em conjunto.

Type: Armazena os tipos de uma vulnerabilidade. Esses tipos estão representados na ontologia como instâncias.

Consequence: Representa informações das possíveis conseqüências da exploração de uma vulnerabilidade. É especializada nas subclasses **LossofFeatures**, **Severity** e **SystemAffected**. Para essas subclasses, suas possibilidades estão representadas como instâncias.

Correction: Essa classe representa informações sobre as correções disponíveis para uma vulnerabilidade cadastrada, código CVE e URL da correção. Essa classe se relaciona com a classe **Supplier** por meio do relacionamento *hasCorrectionSupplier*, indicando que uma correção pode ser disponibilizada por uma ou mais organizações.

Range: Representa informações sobre como a vulnerabilidade pode ser explorada e quais as condições necessárias para que isso ocorra. As possibilidades também são representadas como instâncias.

Product: Representa informações a respeito dos produtos que apresentam vulnerabilidades cadastradas, nome e versão do produto. Possui duas subclasses: Software e Hardware. Essa classe se relaciona com a classe **Supplier** por meio do relacionamento *hasSupplier*, indicando que um produto pode ser fornecido por uma ou mais empresas, e com a classe **Vulnerability** por meio do relacionamento *hasVulnerability*, indicando que um produto pode ter uma ou mais vulnerabilidades.

Supplier: Representa informações referentes às organizações desenvolvedoras e fornecedoras de produtos e correções. Os relacionamentos *developsCorrection* e *sellsProduct* são utilizados para indicar que uma empresa (ou mais) desenvolvem correções e vendem produtos, respectivamente.

Worm: Representa informações que representam os *worms* que podem explorar vulnerabilidades (*exploresVulnerability*).

O desenvolvimento dessa ontologia foi um passo inicial importante para o desenvolvimento da OntoSec. Todos os conceitos, relacionamentos e atributos da Ontologia de Vulnerabilidades descritos nesta seção são aqueles definidos por Brandão quando da definição da ontologia.

3.5 Considerações Finais

Neste capítulo foram apresentados os principais conceitos relacionados à ontologia. Algumas metodologias para desenvolvimento, algumas linguagens de representação e exemplos de ontologias existentes no domínio de segurança foram descritos.

Diversas são as metodologias existentes para o desenvolvimento de ontologias. No entanto, não existe uma que possa ser considerada como um padrão. Geralmente, os pesquisadores que desenvolvem ontologias também desenvolvem sua própria metodologia. Mas, independentemente do domínio de aplicação da ontologia criada, o principal objetivo do desenvolvimento de ontologias é compartilhar terminologia e um conjunto de restrições comuns.

As aplicações de ontologias também são de propósitos variados. De maneira geral, as ontologias são importante para a comunicação, tanto entre pessoas quanto entre sistemas. Na comunicação entre pessoas, elas podem fazer o papel de uma glossário de termos técnicos (ou vocabulário de conceitos); na comunicação entre pessoas e sistemas, elas auxiliam na busca de informação e conhecimento. Já na comunicação entre sistemas, a ontologia define um modelo comum na troca de mensagens (Vieira et al., 2005). As ontologias são valiosas porque representam bem o conhecimento. Mesmo que esse conhecimento seja pouco, ainda é interessante usar ontologias, pois elas são bem estruturadas, e o conhecimento é fonte, ele mesmo, de outros conhecimentos por meio de inferências, comparações ou análises evolutivas.

Para a representação de uma ontologia, é preferível e recomendável que uma linguagem semi-formal ou formal seja utilizada para que a corretitude, a não-ambigüidade e a consistência da ontologia sejam garantidas. No entanto, independentemente da metodologia, linguagem, ferramenta ou domínio de conhecimento, alguns aspectos são importantes e devem ser considerados no desenvolvimento de uma ontologia:

- Definição clara e delimitada do domínio de conhecimento representado. Assim, é possível dizer que uma ontologia é orientada a um domínio e não a uma aplicação específica.

- Interação constante com especialistas do domínio, facilitando a aquisição de conhecimento.
- Definição e planejamento das tarefas que devem ser realizadas.
- Utilização de uma linguagem formal (ou semi-formal) para representar a ontologia. No entanto, o conteúdo de uma ontologia é que permite que ela seja útil, independentemente do formato escolhido para representá-la (Raskin et al., 2001).
- Compartilhamento e reúso da ontologia desenvolvida.

Todos os trabalhos apresentados têm importante contribuição para demonstrar como ontologias podem ser utilizadas para apoiar administradores no gerenciamento de segurança. Algumas apóiam o gerenciamento em alto nível com base em políticas de segurança (nível estratégico de uma organização), enquanto outras apóiam o nível operacional com base em informações geradas por ferramentas de segurança. Nesse sentido, a OntoSec tem como objetivo apoiar o gerenciamento de segurança em um nível gerencial, permitindo que os administradores concentrem em uma única ferramenta todas as informações de incidentes de segurança geradas por diferentes ferramentas de segurança, sejam elas IDSs, *firewalls*, antivírus e outras.

A seguir, o próximo capítulo descreve a OntoSec e seu processo de desenvolvimento.

Capítulo

4

O Desenvolvimento da OntoSec

4.1 Considerações Iniciais

Independentemente da metodologia utilizada ou do domínio de conhecimento, o desenvolvimento de uma ontologia é um processo iterativo, dinâmico e custoso (Das et al., 2001). Uma ontologia evolui constantemente, principalmente para atender às mudanças ocorridas com o domínio de conhecimento que ela representa. Alguns domínios mudam mais rapidamente que outros, mas, geralmente, a primeira versão de uma ontologia sempre muda. Domínios de conhecimento como Medicina, Biologia Molecular e Segurança Computacional são domínios em constante mudança, com novas doenças descobertas, novos tipos de moléculas ou novos tipos de incidentes de segurança, respectivamente.

As razões pelas quais uma ontologia muda estão inerentemente relacionadas com a natureza do domínio de conhecimento, com a habilidade do ser humano em interpretar esse domínio e com o ambiente no qual a ontologia está inserida. Como uma ontologia é a visão do ser humano de um domínio de conhecimento (o especialista), se o ser humano muda sua interpretação a respeito de um determinado conceito ou relacionamento, a ontologia deve representar essa mudança. Nesse sentido, é importante dizer que nenhuma ontologia é 100% completa ou está errada, mas ela deve sim validar os requisitos previamente estabelecidos, ou seja, tudo que o desenvolvedor definiu como conceitos e relacionamentos devem estar representados corretamente. No entanto, mudanças no ambiente podem tornar a ontologia inconsistente, invalidando relacionamentos ou conceitos que deixam

de existir no mundo real. Nesse caso, a ontologia deve ser adaptada para representar as mudanças.

Este capítulo apresenta as atividades realizadas durante o processo de desenvolvimento da OntoSec, os principais conceitos e relacionamentos da ontologia, juntamente com as principais propriedades e restrições de cada conceito. A integração da OntoSec com a Ontologia de Vulnerabilidades (Brandão, 2004) também é descrita.

4.2 Desenvolvimento da OntoSec

Nesta seção são descritas as principais atividades realizadas durante o desenvolvimento da OntoSec de acordo com as metodologias de Fernández et al. (1997) e Noy e McGuinness (2001) apresentadas na Seção 3.2.2 do Capítulo 3.

4.2.1 Planejamento e Especificação

Nessas fases, foram definidos o domínio da ontologia, seus principais objetivos, quais seus usuários em potencial e quais as tarefas e recursos necessários para o seu desenvolvimento.

- **Definição do domínio**: Incidentes de Segurança em sistemas computacionais.
- **Definição do objetivo principal**: Estabelecer uma estrutura única para representar informações sobre incidentes de segurança, possibilitando uma correlação entre esses incidentes.
- **Definição dos usuários**: Os potenciais usuários da OntoSec são administradores de segurança e ferramentas de auxílio ao gerenciamento de segurança, como IDSs e *firewalls*.
- **Definição das tarefas**: As principais tarefas realizadas para o desenvolvimento da OntoSec são aquelas descritas neste capítulo e definidas pela `Methontology` e pela metodologia de Noy e McGuinness.
- **Definição dos recursos**: Os recursos necessários para o desenvolvimento da OntoSec são: ferramenta computacional para modelar a ontologia, linguagem para formalizá-la, e recursos humanos para o processo de desenvolvimento.

4.2.2 Aquisição de Conhecimento

Uma vez definidos o domínio de conhecimento e os objetivos da ontologia, foi necessário pesquisar e estudar diversas fontes de incidentes de segurança para adquirir o conhecimento necessário para o desenvolvimento da ontologia. Dois principais dicionários foram utilizados: *Glossary of Computer Security* (NCSC, 1988) e o *RFC 2828: Internet Security Glossary* (Shirey, 2000). Além desses dicionários, a taxonomia de Howard, descrita na Seção 2.4.4 do Capítulo 2, foi utilizada para a concepção da idéia principal da OntoSec, as informações de segurança gerenciadas pelo CSIRT USP[1] foram utilizadas para definir alguns dos tipos de incidentes que a OntoSec representa.

Além desse estudo e do contato com profissionais de segurança da Universidade de São Paulo, é importante também nessa fase definir para "quais perguntas a ontologia deverá prover respostas", ou seja, deve-se definir quais são as **"questões de competência"** (Gruninger e Fox, 1995). Essas questões guiam a escolha dos conceitos que a ontologia irá representar, facilitando a fase de Conceituação. A seguir, são descritas as principais questões de competência definidas em um nível mais alto de abstração, ou seja, sem instâncias específicas. Essas questões contemplam os principais conceitos da ontologia. Para o processo de avaliação da OntoSec, que é apresentado no Capítulo 5, essas questões foram especializadas.

- Quais os tipos mais freqüentes de incidentes?
- Quais foram as conseqüências de um incidente?
- Quais são as pré-condições para que um incidente ocorra?
- Quais ativos do sistema foram alvos de um incidente?
- Quais ativos são mais atacados e sofrem incidentes?
- Quais ferramentas foram utilizadas para executar um ataque?
- Que tipo de acesso o atacante realizou para executar um ataque?
- Quais são os tipos de ataques mais comuns?
- Quais incidentes precederam ou procederam um incidente?

[1] O CSIRT (*Computer Security Incident Response Team*) USP está sediado no CCE (Centro de Computação Eletrônica) no campus de São Paulo da USP (Universidade de São Paulo). Um CSIRT é uma organização responsável por receber, revisar e responder a incidentes de segurança.

- Quais são as correções de uma vulnerabilidade?
- Quais os tipos mais freqüentes de vulnerabilidades?
- Quais são as vulnerabilidades de um ativo?
- Quais vulnerabilidades são exploradas em conjunto com outras?
- Quais vulnerabilidades foram exploradas durante um incidente?
- Quais vulnerabilidades são pré-condições para uma vulnerabilidade?
- Em qual dia da semana ou mês ocorrem mais incidentes?
- Quais portas ou protocolos são mais utilizados para causar incidentes?

A atividade de aquisição de conhecimento foi realizada durante todo o processo de desenvolvimento da OntoSec. E como o domínio de conhecimento de incidentes de segurança é muito dinâmico e muda rapidamente, realizar essa atividade constantemente foi crucial.

4.2.3 Conceituação

A fase de conceituação de uma ontologia é a mais difícil e trabalhosa. Definir quais conceitos serão representados e quais não serão depende muito do conhecimento do domínio de conhecimento. Por isso, o uso de dicionários, glossários, artigos e o contato com profissionais da área são importantes. A partir desse estudo, dos objetivos definidos na fase de Planejamento e Especificação, e das questões de competências definidas, os conceitos da OntoSec foram definidos.

A idéia principal representada na OntoSec é a seguinte: um **Agente** *executa* um **Ataque** que pode *causar* um **Incidente de Segurança**. Para executar ataques, um agente pode, *utilizando* uma **Ferramenta**, *explorar* uma **Vulnerabilidade** e *adquirir* **Acesso** a um sistema. Um incidente de segurança, por sua vez, *implica em* uma **Conseqüência**, *age em* um **Ativo** e *possui* uma **Pré-Condição**. Essa pré-condição pode *estar relacionada com* uma vulnerabilidade. Uma conseqüência pode ainda *estar relacionada a* um ativo. Além disso, um incidente de segurança pode *preceder* e/ou *proceder* outros incidentes. Já uma vulnerabilidade *ocorre* em um ativo, tem uma **Abrangência**, um **Tipo** e uma **Correção**. Essa correção é *desenvolvida* por uma **Organização**[2]. Uma

[2]Na ontologia, o conceito organização é utilizado como sinônimo de empresa.

vulnerabilidade pode ainda se relacionar com outras vulnerabilidades quando uma vulnerabilidade é *pré-condição* para que outra seja explorada, ou uma vulnerabilidade é *explorada em conjunto* com outras[3]. Essa idéia, ilustrada na Figura 4.1, representa os conceitos de mais alto nível do domínio de incidentes de segurança e como eles se relacionam. Eles compõem o *core* da OntoSec.

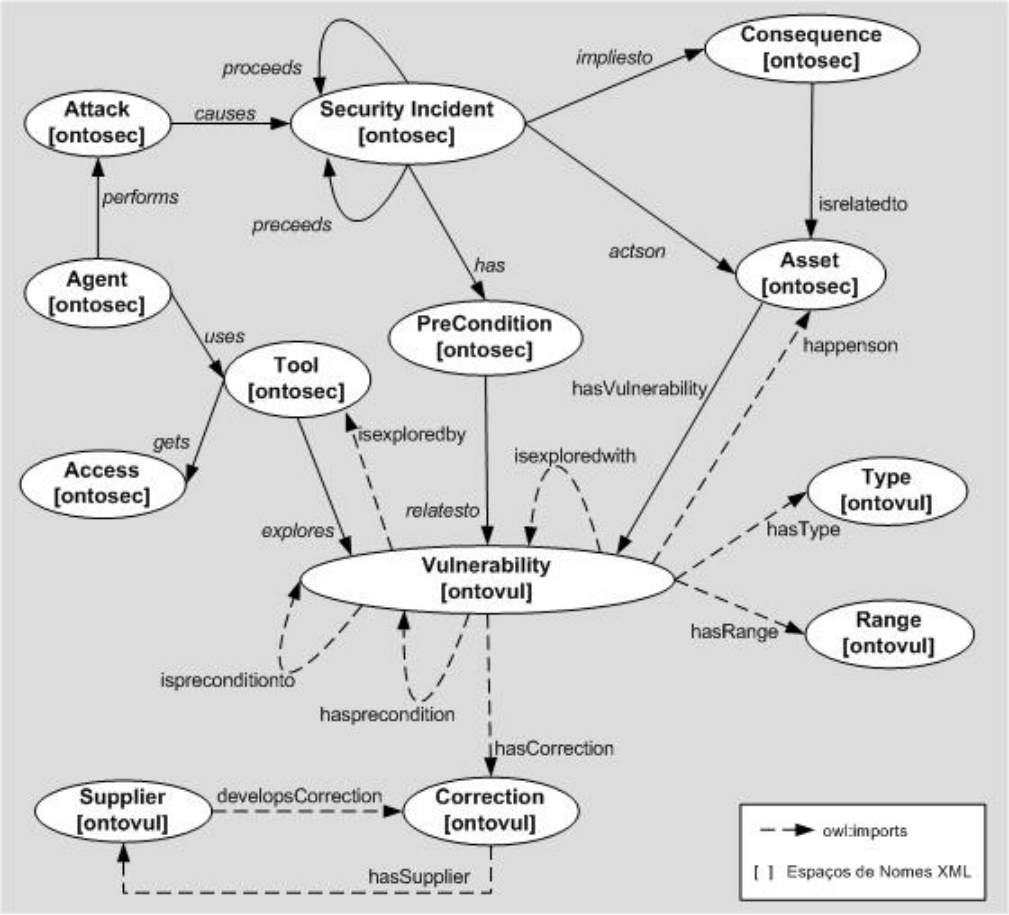

Figura 4.1: Principais conceitos e relacionamentos da OntoSec.

A partir dos conceitos de alto nível, a OntoSec foi especializada. A abordagem utilizada para esse processo é a *middle-out*, proposta por Uschold e Gruninger (1996). Essa abordagem identifica os principais conceitos de um domínio de conhecimento a partir dos quais a ontologia é desenvolvida. A vantagem em se utilizar essa abordagem ao invés das abordagens *top-down* e *bottom-up*, também propostas por Uschold e Gruninger, é que uma vez que os conceitos principais estão definidos e são estáveis, é possível

[3]O espaço de nomes XML `ontovul` identifica os conceitos que foram importados da Ontologia de Vulnerabilidades.

especializá-los quando necessário, minimizando o esforço de manutenção da ontologia e criando a hierarquia de classes, além dos relacionamentos não-hierárquicos.

A seguir, para cada um dos conceitos são apresentadas sua especialização (quando existir), suas propriedades (atributos e relacionamentos não-hierárquicos) e suas restrições[4]. Como recomenda Noy e McGuinness (2001), a maior parte das classes foram especializadas quando a subclasse criada possuía propriedades além daquelas herdadas da superclasse.

A OntoSec possui 49 conceitos representados como classes, sendo 13 no nível zero (0), 11 no nível um (1), 14 no nível dois (2) e 11 no nível três (3), como mostra a Figura 4.2[5]. Além desses conceitos, a OntoSec possui ainda outros conceitos que estão representados como atributos ou instâncias, e 36 relacionamentos não-hierárquicos. O Apêndice A apresenta o vocabulário de conceitos e o vocabulário de relacionamentos da OntoSec.

Classe SecurityIncident. Essa classe representa o principal conceito da OntoSec, que é o "incidente de segurança". Para essa classe foram definidas propriedades que armazenam informações referentes aos incidentes. A classe **SecurityIncident** possui as propriedades descritas a seguir e não possui nenhuma especialização.

Atributos:

- ***has_date***: Data (ano, mês e dia) na qual um incidente ocorreu. Um incidente possui uma única data, sendo, assim, uma propriedade funcional.
- ***has_time***: Hora (horas, minutos e segundos) na qual um incidente ocorreu. Um incidente possui uma única hora.
- ***has_description***: Descrição única do incidente.
- ***has_reference***: Referências contendo mais detalhes sobre um incidente, como *sites* ou boletins de segurança.
- ***has_severity***: Define o grau de gravidade do incidente, podendo ser ou *Low*, ou *Medium* ou *High*. Também é uma propriedade funcional.
- ***has_weekday***: Dia da semana no qual ocorreu um incidente. Também é uma propriedade funcional.
- ***hasip_destination***: Endereços IPs dos computadores alvos de um incidente.
- ***hasip_source***: Endereços IPs dos computadores que iniciaram um incidente.

[4]Na descrição da OntoSec, os principais conceitos são referenciados como classes.

[5]Figura gerada com o *plugin* ***OWLViz***. A classe **Thing** representa uma classe abstrata a partir da qual as outras classes são criadas. Essa classe é criada por *default* na ferramenta Protégé.

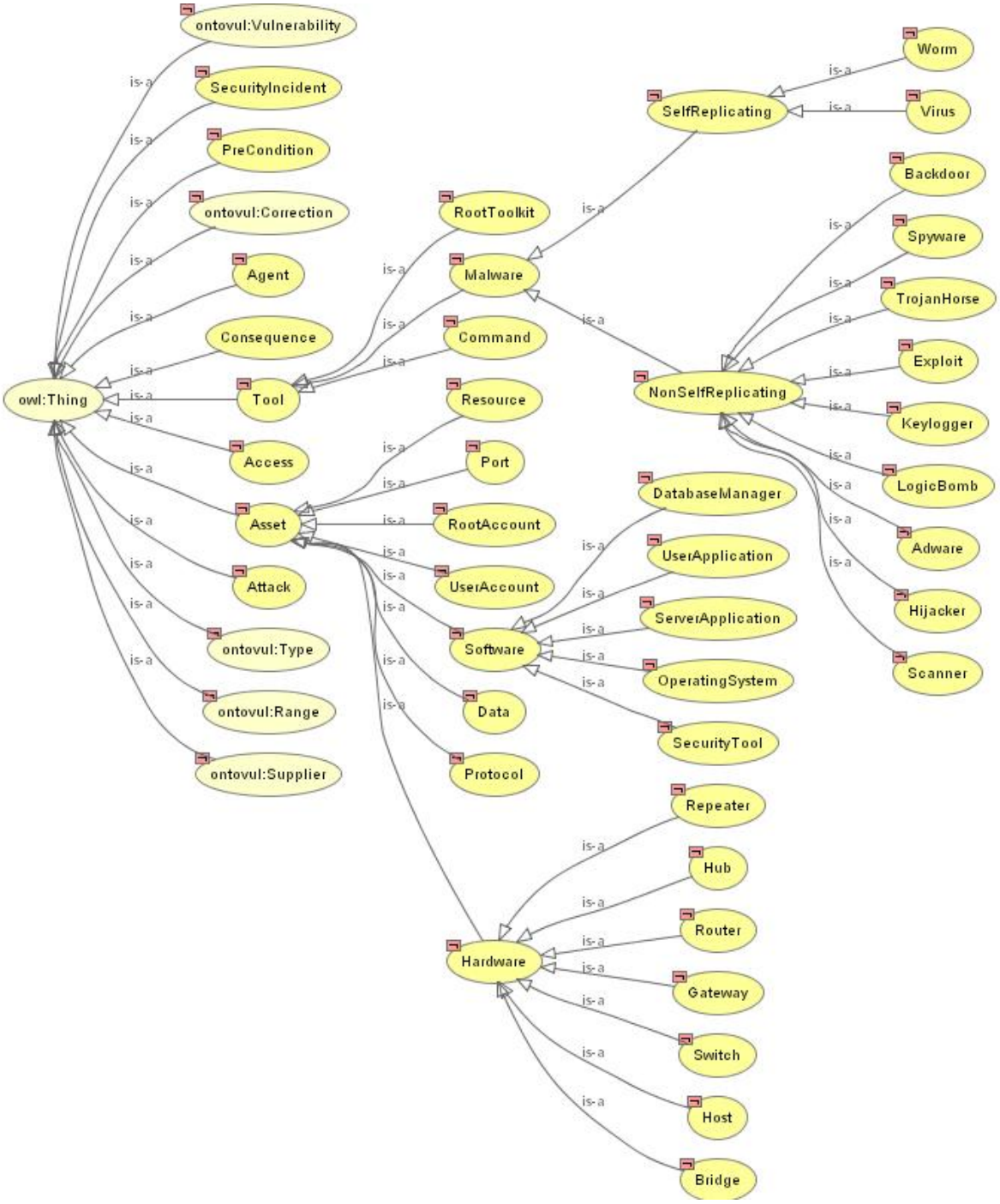

Figura 4.2: Hierarquia de classes da OntoSec.

- ***hassecurity_incident_type***: Identifica os tipos que um incidente pode ter, podendo ser mais de um. Essa propriedade possui instâncias predefinidas com os tipos de incidentes de segurança (conceitos) mais freqüentemente relatados. São eles: *AccountCompromise*, *BufferOverflow*, *CookiePoisoning*, *CrossSi-*

teScripting, *Deface*, *DenialOfService*, *DistributedDenialOfService*, *Harvesting*, *Hijack*, *MaliciousCode*, *ManIntheMiddle*, *MiscellaneousAttack*, *Misuse*, *NaturalDisaster*, *OpenRelay*, *OpenProxy*, *PhysicalProblem*, *Pornography*, *Probe*, *RootCompromise*, *ScamPhishing*, *Scanning*, *Sniffing*, *SocialEngineering*, *Spam*, *Spoofing* e *SQLInjection*.

Relacionamentos não-hierárquicos:

- *actson_Asset*: Indica que um incidente atinge um ou mais alvos (ou ativos), como por exemplo, um servidor de banco de dados, um servidor Web, um sistema operacional, um roteador ou uma conta de usuário.
- *impliesto_Consequence*: Indica as conseqüências que um incidente pode causar no sistema.
- *precedes_SecurityIncident*: Indica se um ou mais incidentes precedem outros. É inverso ao relacionamento *proceeds_SecurityIncident*.
- *proceeds_SecurityIncident*: Indica se um ou mais incidentes procedem outros.
- *has_PreConditions*: Representa as condições que um sistema deve ter para possibilitar que um incidente ocorra, como por exemplo, a presença de uma vulnerabilidade no sistema operacional ou senhas fáceis. Essa propriedade tem como valores as instâncias da classe **PreCondition**.
- *uses_Port*: Indica as portas destino e origem da classe **Port** que são utilizadas durante um incidente.
- *uses_Protocol*: Indica os protocolos da classe **Protocol** que são utilizados para causar um incidente.

Classe Agent. Essa classe identifica o agente que executou um ou mais ataques que causam incidentes de segurança. Ela possui os seguintes atributos e relacionamentos:

Atributos:

- ***hasagent_type***: Identifica o tipo de agente que realiza ataques e causam incidentes. Possui como instâncias predefinidas: *Human*, que representa os agentes humanos, como os *crackers*, *Natural*, que representa os agentes da natureza, como uma inundação, e *Physical*, que representa os agentes físicos, como um disco rígido com defeito ou uma falha elétrica.

Relacionamentos não-hierárquicos:

- *performs_Attack*: Indica que um agente executa ataques. Esse relacionamento pode ter cardinalidade "zero"(0) quando o agente que causou o incidente é do tipo *Natural* ou *Physical*.
- *uses_Tool*: Indica que um agente pode utilizar uma ou mais ferramentas para executar ataques. Essas ferramentas são representadas pela classe **Tool**, e são códigos maliciosos, pacotes de software ou comandos.

Classe Attack. Essa classe identifica os ataques executados pelos agentes que causam incidentes de segurança. Ela possui os seguintes atributos e relacionamentos:

Atributos:

- ***hasattack_location***: Identifica a localização do atacante. Tem como instâncias predefinidas: *Local* e *Remote*, que representam, respectivamente, ataques que ocorrem de dentro do sistema e ataques que ocorrem de fora do sistema. As instâncias são exclusivas, ou seja, ou o ataque é local ou remoto.
- ***hasattack_type***: Identifica o tipo de ataque. Tem como instâncias predefinidas: *Passive* e *Active*, que representam, respectivamente, ataques passivos nos quais o atacante apenas observa o sistema e ataques ativos nos quais o atacante causa danos ao sistema. As instâncias são exclusivas, ou o ataque é passivo ou ativo.

Relacionamentos não-hierárquicos:

- *causes_SecurityIncident*: Indica os incidentes de segurança causados por um ataque.

Classe Access. Essa classe identifica o tipo de acesso que um atacante adquire. Essa classe não possui nenhum relacionamento e possui os seguintes atributos:

- ***hasaccess_type***: Identifica o tipo de acesso. Tem como instâncias predefinidas: *RootAccess* e *UserAccess*, que representam, respectivamente, acesso à conta de administrador do sistema (ou *root*) e acesso a uma conta de usuário comum do sistema. As instâncias são exclusivas.
- ***hasuser_type***: Identifica o tipo de usuário utilizado para acesso. Tem como instâncias predefinidas: *AuthorizedUser* e *UnauthorizedUser*, que representam, respectivamente, acessos com usuários autorizados (legítimos no sistema) e acessos com usuários não autorizados (não legítimos no sistema). As instâncias são exclusivas.

Classe PreCondition. Essa classe identifica as pré-condições necessárias para a ocorrência de um incidente de segurança. Uma vez que uma pré-condição pode ser uma vulnerabilidade, essa classe possui o relacionamento *relatesto_Vulnerability* com a classe **Vulnerability**. A classe possui apenas o atributo ***hasprecondition_description***, que descreve a pré-condição propriamente dita.

Classe Consequence. Essa classe representa as conseqüências que um incidente pode causar no sistema por meio do atributo ***has_consequence***, cujo valores estão descritos a seguir. Ela possui um único relacionamento com a classe **Asset**, o *relatesto_Asset*, que permite identificar que conseqüências um ativo pode ter. Por exemplo, a conseqüência **TheftofResource** está relacionada com o ativo **Resource**, ou a conseqüência **LossofPrivacy** está relacionada com o ativo **UserAccount** ou **RootAccount**.

Os tipos de conseqüências foram definidos com base nos tipos de incidentes de segurança e nas principais características que um sistema seguro deve apresentar, que são: disponibilidade, integridade, confidencialidade e autenticidade. As conseqüências são:

- *ConfigurationChange*: Identifica incidentes que modificam configurações do sistema, como por exemplo, atribuir privilégios não autorizados a usuários mal-intencionados ou abrir portas para futuras conexões.
- *LossAvailability*: Identifica incidentes que causam indisponibilidade do sistema ou de seus dados e informações.
- *LossIntegrity*: Identifica incidentes que causam mudanças não autorizadas no sistema ou em seus dados e informações.
- *LossConfidentiality*: Identifica incidentes que causam exposição não autorizada do sistema ou de seus dados e informações.
- *LossofPrivacy*: Identifica incidentes que expõem a conta de um usuário legítimo do sistema, seja usuário comum ou administrador.
- *NaturalDamage*: Identifica incidentes que danificam fisicamente algum componente do sistema devido a atos da natureza. Representa problemas como inundações.
- *PhysicalDamage*: Identifica incidentes que danificam fisicamente algum componente do sistema devido a defeitos de hardware. Representa problemas como defeitos em discos rígidos ou falhas elétricas.

- *RemoteExecution*: Identifica incidentes que permitem um acesso e execução de comandos remotos ao sistema.
- *TheftofResource*: Identifica incidentes que causam roubo de recursos do sistema, tais como: processamento, memória, largura de banda, serviço ou dispositivos de entrada/saída (E/S).

Classe Asset. Essa classe representa os ativos de um sistema que podem ser alvos de incidentes de segurança. Ela não possui atributos, e possui um único relacionamento com a classe **Vulnerability**, o *hasVulnerability*, que identifica quais vulnerabilidades um ativo pode ter. Esse relacionamento é inverso ao relacionamento *happenson_Asset* da classe **Vulnerability**.

A Figura 4.3 mostra a hierarquia de classes da classe **Asset** e o significado de cada subclasse é descrito a seguir. Essa especialização está baseada nos principais ativos que uma organização tem disponível para que seus usuários executem suas tarefas. Para as subclasses **Hardware** e **Software**, que possuem diversos atributos e relacionamentos, uma descrição mais detalhada é apresentada.

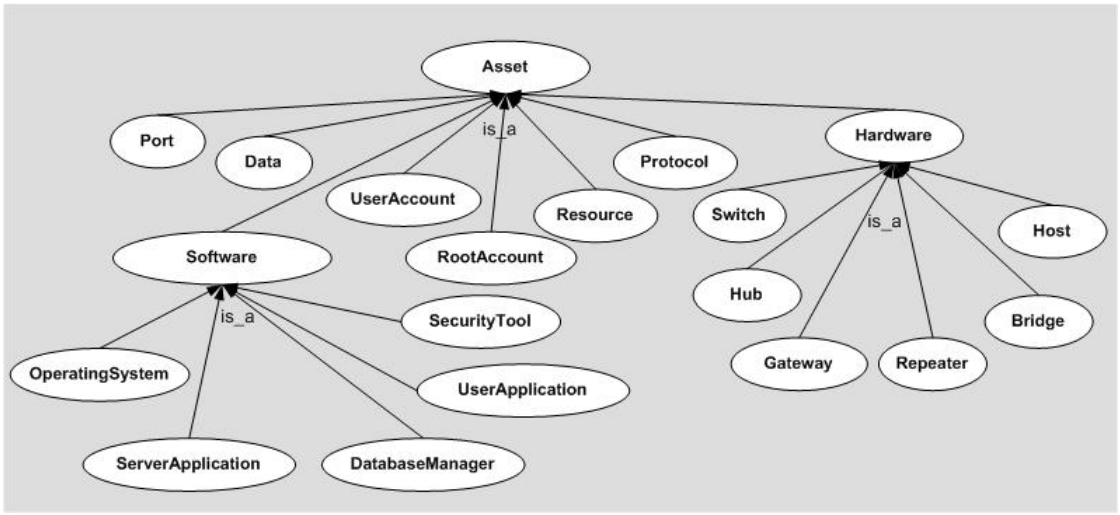

Figura 4.3: Hierarquia de classes da classe **Asset**.

Hierarquia de Classes:

- **Data**: Representa qualquer dado (ou arquivo) que seja alvo de incidentes.
- **Hardware**: Representa os equipamentos que podem ser alvos de incidentes. Essa classe possui como subclasses: **Bridge**, **Gateway**, **Host**, **Hub**, **Repeater**, **Router**, **Switch**.

- **Port**: Representa as portas lógicas que podem ser ou alvos de um incidente ou utilizadas durante um incidente. A classe possui os seguintes atributos: ***has_destinyportnumber*** e ***has_sourceportnumber***, que são porta destino e porta origem, respectivamente. Essa classe se relaciona com a classe **SecurityIncident** por meio do relacionamento *isusedby_SecurityIncident*, indicando que uma ou mais portas podem ser utilizadas por um incidente de segurança. Esse relacionamento é inverso ao relacionamento *uses_Port* da classe **SecurityIncident**.

- **Protocol**: Representa os protocolos que podem ser ou alvos de um incidente ou utilizados durante um incidente. A classe possui como atributos: ***hasprotocol_name*** e ***hasprotocol_version***, nome e versão do protocolo, respectivamente. O atributo ***hasprotocol_name*** possui instâncias pré-definidas: ARP (*Address Resolution Protocol*), DHCP (*Dynamic Host Configuration Protocol*), FTP, HTTP (*Hypertext Transfer Protocol*), HTTPS (*Secure Hypertext Transfer Protocol*), ICMP, IMAP (*Internet Message Access Protocol*), IP, LDAP (*Lightweight Directory Access Protocol*), NetBios (*Network Basic Input/Output System*), POP (*Post Office Protocol*), SFTP(*Secure FTP*), SMTP (*Simple Mail Transfer Protocol*), SNMP (*Simple Network Management Protocol*), SSH (*Secure Shell*), SSL (*Secure Socket Layer*), TCP, Telnet (*Telephone Network*), TLS (*Transport Layer Security*) e UDP. Essas instâncias representam os principais protocolos (conceitos) utilizados atualmente pelas redes de computadores. A classe se relaciona com a classe **SecurityIncident** por meio do relacionamento *usedby_SecurityIncident*, indicando que um ou mais protocolos podem ser utilizados durante um incidente de segurança. Esse relacionamento é inverso ao relacionamento *uses_Protocol* da classe **SecurityIncident**.

- **Resource**: Representa os recursos que podem ser alvos de um incidente. A classe possui o seguinte atributo: ***hasresource_type***, que tem como instâncias predefinidas os valores: *Bandwidth*, *CommunicationChannel*, *IODevice*, *Memory*, *Processor* e *Service*.

- **Software**: Representa os programas que podem ser alvos de um incidente. A classe possui como subclasses: **DatabaseManager**, **OperatingSystem**, **SecurityTool**, **ServerApplication**, **UserApplication**.

- **UserAccount**: Representa uma conta de usuário que pode ser alvo de um incidente. A classe possui o atributo ***has_userid***, que identifica a conta de maneira única.

- **RootAccount**: Representa uma conta de administrador que pode ser alvo de um incidente. A classe possui o atributo ***has_rootid***, que identifica a conta de maneira única.

Classe Hardware. Além das propriedades herdadas da classe **Asset**, essa classe possui os seguintes atributos e relacionamentos:

Atributos:

- ***has_ip***: Identifica o endereço IP do hardware, caso exista. Um hardware pode ter mais de um endereço IP.
- ***hasmodel_hw***: Modelo do hardware.
- ***hasname_hw***: Nome do hardware, caso exista.

Relacionamentos não-hierárquicos:

- *hasSupplier*: Identifica a organização da classe **Supplier** que desenvolve ou fornece o hardware. Esse relacionamento é inverso ao relacionamento *sellsProduct* da classe **Supplier**.

Classe Software. Além das propriedades herdadas da classe **Asset**, essa classe possui os seguintes atributos e relacionamentos:

Atributos:

- ***hasname_sw***: Nome do software.
- ***hasversion_sw***: Versões do software.

Relacionamentos não-hierárquicos:

- *hasSupplier*: Identifica a organização da classe **Supplier** que desenvolve ou fornece o software. Esse relacionamento é inverso ao relacionamento *sellsProduct* da classe **Supplier**.
- *usesPort*: Identifica quais portas da classe **Port** são utilizadas pelo software para atender solicitações de serviços.
- *usesProtocol*: Identifica quais protocolos da classe **Protocol** são utilizados pelo software durante sua execução.

Além dos atributos da classe **Software**, as subclasses **SecurityTool** e **ServerApplication** possuem atributos extras que identificam, respectivamente, o tipo de

uma ferramenta de segurança e o tipo de servidor, que são: ***has_sectooltype*** e ***has_servertype***. O atributo ***has_sectooltype*** tem como instâncias predefinidas os valores: *Antivirus*, *CryptographyModule*, *Firewall*, *IntrusionDectectionSystem*, e *Proxy*, e o atributo ***has_servertype*** tem como instâncias predefinidas os valores: *DNSServer (Domain Name Service)*, *FileServer*, *FileTransferServer*, *MailServer*, *NISServer (Network Information Service)*, *PrintServer*, *WebServer* e *WINSServer (Windows Internet Naming Service)*. Essas instâncias representam as principais ferramentas de gerenciamento de segurança e os principais serviços utilizados e oferecidos por uma organização, respectivamente.

Classe Tool. Essa classe representa os mecanismos ou ferramentas que um atacante pode utilizar para causar incidentes de segurança. A classe possui os seguintes atributos e relacionamentos:

Atributos:

- ***hasdate_tool***: Data de criação da ferramenta.
- ***hasname_tool***: Nome da ferramenta.
- ***hasreference_tool***: *Sites* com referências sobre a ferramenta.
- ***hasversion_tool***: Versão da ferramenta.

Relacionamentos não-hierárquicos:

- *explores_Vulnerability*: Indica que uma ferramenta pode explorar uma ou mais vulnerabilidades para causar incidentes de segurança.
- *gets_Access*: Indica que uma ferramenta permite acesso ao sistema e esse acesso pode causar incidentes de segurança.

A classe **Tool** possui uma hierarquia de classes, como mostra a Figura 4.4. Essa hierarquia está baseada também nos seguintes glossários de códigos maliciosos: *Malicious Codes in Depth* (Heidari, 2004) e *Glossary of Virus Terms* (TrendMicro, 2006).

- **RootToolkit**: Pacotes de software que podem ser utilizados pelos atacantes. Além dos atributos herdados da classe **Tool**, essa classe possui o atributo ***is_package***, que é definido como "booleano" e tem valor fixo *true*.
- **Command**: Comandos utilizados pelos atacantes. Além dos atributos herdados da classe **Tool**, essa classe possui o atributo ***uses_commandline***, que também é definido como "booleano" e tem valor fixo *true*.

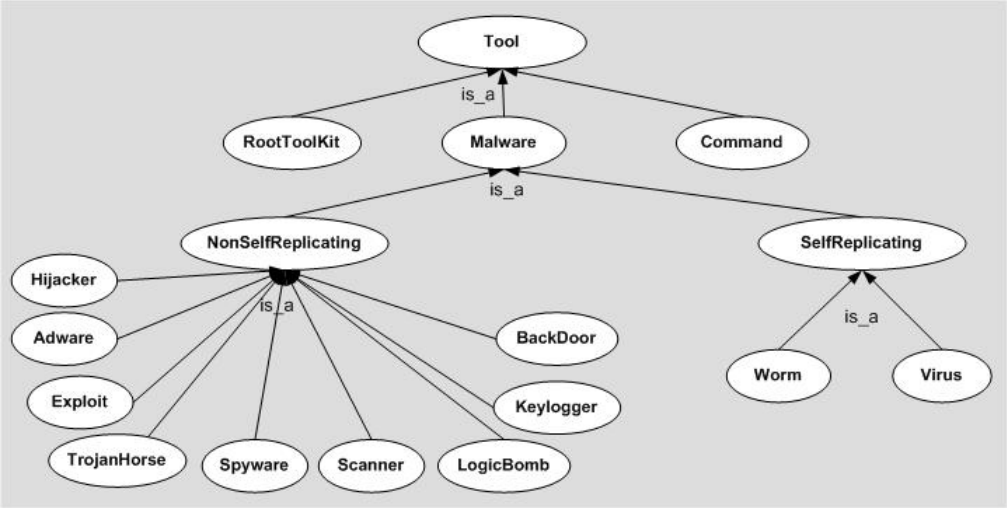

Figura 4.4: Hierarquia de classes da classe **Tool**.

- **Malware**: Códigos maliciosos utilizados pelos atacantes. Essa classe possui duas subclasses que separam os programas maliciosos entre aqueles que têm a capacidade de se auto replicarem e aqueles que não têm, que são as classes **SelfReplicating** e **NonSelfReplicating**, respectivamente. Essas classes possuem as seguintes subclasses:
  - **SelfReplicating**: **Virus** e **Worm**. Para diferenciar um vírus de um *worm* (e vice-versa), a classe **SelfReplicating** possui o atributo ***user_interaction***. Esse atributo é do tipo "booleano" e indica se o código malicioso necessita da intervenção do usuário para se replicar ou não. Para a classe **Virus** seu valor é *true* e para a classe **Worm** seu valor é *false*.
  - **NonSelfReplicating**: **Adware**, **BackDoor**, **Exploit**, **Hijacker**, **Keylogger**, **LogicBomb**, **Scanner**, **Spyware** e **TrojanHorse**. Cada uma dessas subclasses possui um atributo que as diferencia, com exceção das classes **Exploit** e **LogicBomb**. No caso da classe **LogicBomb**, ela possui um relacionamento com a classe **PreCondition**, *hasPrecondition*, indicando que esse código malicioso precisa de pré-condições para ser executado no sistema. Todos os atributos são booleanos com valor predefinido *true*.
    - **Adware**: atributo ***hasmeans_browser***, que indica um código malicioso que ataca um navegador Web.
    - **Backdoor**: atributo ***opens_breach***, que indica um código malicioso que abre brechas no sistema que podem ser utilizadas para causar incidentes de segurança. Essa classe também possui um relacionamento

com a classe **TrojanHorse**, *relatesto_TrojanHorse*, uma vez que um *trojan horse* pode esconder um *backdoor*.

- **Hijacker**: atributo ***modifies_site***, que indica um código malicioso que modifica um *site*.
- **Keylogger**: atributo ***hasmeans_keyboard***, que indica um código malicioso que utiliza o teclado para coletar informações e posteriormente causar incidentes.
- **Scanner**: atributo ***explores_system***, que indica um código malicioso que explora o sistema em buscas de falhas ou vulnerabilidades.
- **Spyware**: atributo ***gathers_information***, que indica um código malicioso que coleta informações do sistema que serão posteriormente utilizadas para causar incidentes. Essa classe também possui um relacionamento com a classe **TrojanHorse**, *relatesto_THorse*, uma vez que um *trojan horse* pode esconder um *spyware*.
- **TrojanHorse**: atributo ***looks_harmless***, que indica que esse tipo de código malicioso parece inocente ao olhos do usuário mas não é. Essa classe também possui dois relacionamentos, um com a classe **Backdoor** e outro com a classe **Spyware**, *relatesto_Backdoor* e *relatesto_Spyware*, respectivamente.

Classe Vulnerability. Essa classe armazena as vulnerabilidades que um ativo do sistema pode ter, e possui os seguintes atributos e relacionamentos:

Atributos:

- ***has_cvecode***: Identifica unicamente a vulnerabilidade por meio do código CVE.
- ***description***: Descreve a vulnerabilidade (descrição do CVE).
- ***reference***: *Sites* ou boletins com descrições mais detalhadas sobre a vulnerabilidade.
- ***publish_date***: Data de divulgação da vulnerabilidade.

Relacionamentos não-hierárquicos:

- *hasType*: Identifica o tipo de uma vulnerabilidade.
- *hasRange*: Identifica como uma vulnerabilidade pode ser explorada.
- *hasCorrection*: Identifica as correções de uma vulnerabilidade.

- *hasPreCondition*: Identifica quais vulnerabilidades são pré-condição para que uma vulnerabilidade exista no sistema e seja explorada. É inverso ao relacionamento *preconditionto*.
- *preconditionto*: Identifica para quais vulnerabilidades uma vulnerabilidade é pré-condição. É inverso ao relacionamento *hasPreCondition*.
- *exploredwithVulnerability*: Identifica quais vulnerabilidades podem ser exploradas em conjunto.
- *isexploredby_Tool*: Identifica quais ferramentas exploram vulnerabilidades.
- *happenson_Asset*: Identifica quais vulnerabilidades os ativos da classe **Asset** têm. Esse relacionamento foi criado após a integração com a Ontologia de Vulnerabilidades e é inverso ao relacionamento *hasVulnerability* da classe **Asset**.

Classe Range. Essa classe armazena informações sobre como uma vulnerabilidade pode ser explorada. Ela possui um único atributo, o ***has_range***, que tem como instâncias predefinidas os valores: *Interception*, *ClientSide* e *PhysicalAccess*. Esse atributo foi criado após a integração com a Ontologia de Vulnerabilidades.

Classe Type. Essa classe armazena os tipos de uma vulnerabilidade. Esses tipos estão representados pelo atributo ***has_vultype***, que possui como instâncias predefinidas os valores: *AccessValidationError*, *ConfigurationError*, *ContextError*, *DesignError*, *ExceptionError* e *RaceCondition*. Esse atributo foi criado após a integração com a Ontologia de Vulnerabilidades.

Classe Correction. Essa classe armazena informações sobre as correções disponíveis para uma vulnerabilidade. Ela possui dois atributos: ***cvecode*** e ***url***, que informam, respectivamente, o código CVE da correção e o(s) *site(s)* onde a correção pode ser encontrada. Essa classe se relaciona com a classe **Supplier** por meio do relacionamento *hasCorrectionSupplier*, indicando que uma correção pode ser disponibilizada por uma ou mais organizações.

ClasseSupplier. Essa classe armazena informações referentes às organizações que desenvolvem ou fornecem tanto produtos de software e hardware quanto correções para vulnerabilidades. Ela possui os seguintes atributos e relacionamentos:

Atributos:

- ***name***: Nome da organização.
- ***email_contact***: E-mails de contato.

- ***phone_contact***: Telefones de contato.
- ***website***: *Sites* da organização.

Relacionamentos não-hierárquicos:

- *developsCorrection*: Indica que uma organização desenvolve correções para vulnerabilidades. É inverso ao relacionamento *hasCorrectionSupplier* da classe **Correction**.
- *sellsProduct*: Indica que uma organização desenvolve ou fornece produtos de software e hardware (classes **Software** e **Hardware**). Esse relacionamento foi modificado após a integração da OntoSec com a Ontologia de Vulnerabilidades, pois a classe **Product** não foi importada da Ontologia de Vulnerabilidades, e ele é inverso ao relacionamento *hasSupplier* das classes **Software** e **Hardware**.

4.2.4 Formalização

A OntoSec foi formalizada utilizando a linguagem *OWL DL* descrita na Seção 3.2.3 do Capítulo 3, e modelada com a ferramenta Protégé, por meio do editor ***Protégé-OWL***, que permite gerar automaticamente o código OWL da ontologia.

O código a seguir mostra o cabeçalho da OntoSec:

```
1 <?xml version="1.0"?>
2 <rdf:RDF
3    xmlns:jms="http://jena.hpl.hp.com/2003/08/jms#"
4    xmlns="http://a.com/SecurityIncidentOntology#"
5    xmlns:owl="http://www.w3.org/2002/07/owl#"
6    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
7    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
8    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
9    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
10   xmlns:ontovul="http://a.com/OntoVulnerability#"
11   xml:base="http://a.com/SecurityIncidentOntology">
12   <owl:Ontology rdf:about="">
13     <owl:imports rdf:resource="http://a.com/OntoVul.owl"/>
14   </owl:Ontology>
```

As duas primeiras declarações (linhas 1 e 2) identificam o espaço de nomes XML e RDF associados à ontologia. A linha 3 identifica o *framework* Jena, que é utilizado pela ferramenta Protégé. As linhas 4 e 11 identificam o espaço de nomes da ontologia, `SecurityIncidentOntology`. As declarações das linhas 5 a 9 são de espaços de nomes e indicam que elementos com os prefixos `owl`, `rdf`, `rdfs`, `xsd` e `daml` devem ser entendidos

como definições pertencentes aos vocabulários das linguagens OWL, RDF, Esquema RDF, Esquema XML e DAML+OIL, respectivamente. A declaração da linha 10 mostra que a Ontologia de Vulnerabilidades é identificada com o rótulo `ontovul`, enquanto que as linhas 12 a 14 mostram que a ontologia é importada do arquivo *OntoVul.owl* com o construtor `owl:imports`.

Toda a expressividade da *OWL DL* foi utilizada para formalizar a OntoSec. No entanto, são apresentados apenas alguns exemplos da definição de classes, subclasses, atributos, relacionamentos e restrições.

Para definição das classes e subclasses, os construtores owl:Class e rdfs:subClassOf foram utilizados, respectivamente. O trecho de código a seguir mostra as definições das classes **SecurityIncident**, **Consequence**, **Asset**, **Tool**, e **Vulnerability** (linhas 1 a 5), e da subclasse **RootAccount** (linhas 7 a 9) como exemplos.

```
1 <owl:Class rdf:ID="SecurityIncident"/>
2 <owl:Class rdf:ID="Consequence"/>
3 <owl:Class rdf:ID="Asset"/>
4 <owl:Class rdf:ID="Tool"/>
5 <owl:Class rdf:ID="http://a.com/OntoVulnerability#Vulnerability"/>
6
7  <owl:Class rdf:ID="RootAccount">
8    <rdfs:subClassOf rdf:resource="#Asset"/>
9 </owl:Class>
```

Para definição de atributos de uma classe, o construtor owl:DatatypeProperty foi utilizado. Esse construtor permite que restrições sejam estabelecidas, tais como: número de instâncias que um atributo pode ter, cardinalidade, instâncias pré-definidas para um atributo (construtores owl:DataRange e owl:oneof), e também definir que um atributo pode possuir apenas um valor, ou seja, ser funcional (construtor owl:FunctionalProperty). O trecho de código a seguir mostra como alguns dos atributos da classe **SecurityIncident** foram definidos.

O atributo ***has_date*** é uma propriedade funcional do tipo "XMLSchema#date", pois um determinado incidente de segurança tem apenas uma data associada a ele (linhas 1 a 5). O atributo ***has_severity*** é do tipo "XMLSchema#string" e possui instâncias pré-definidas: *Low* (linha 14), *Medium* (linha 17) e *High* (linha 20). Como um incidente só pode ter apenas uma dessas instâncias, esse atributo também é funcional (linha 8). O atributo ***hasip_destination*** é do tipo "XMLSchema#string" e possui mínimo de cardinalidade igual a um (1), indicando que um incidente de segurança pode ter mais de um IP destino associado a ele. Todos os tipos de um atributo da OntoSec são definidos pelo Esquema XML.

```
1 <owl:FunctionalProperty rdf:ID="has_date">
2   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
3    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
4    <rdfs:domain rdf:resource="#SecurityIncident"/>
5 </owl:FunctionalProperty>
6
7 <owl:DatatypeProperty rdf:ID="has_severity">
8    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
9    <rdfs:domain rdf:resource="#SecurityIncident"/>
10    <rdfs:range>
11      <owl:DataRange>
12        <owl:oneOf rdf:parseType="Resource">
13          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
14          >Low</rdf:first>
15          <rdf:rest rdf:parseType="Resource">
16            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
17            >Medium</rdf:first>
18            <rdf:rest rdf:parseType="Resource">
19              <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
20              >High</rdf:first>
21              <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
22            </rdf:rest>
23          </rdf:rest>
24        </owl:oneOf>
25      </owl:DataRange>
26    </rdfs:range>
27 </owl:DatatypeProperty>
28
29 <owl:Restriction>
30   <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
31    >1</owl:minCardinality>
32    <owl:onProperty>
33      <owl:DatatypeProperty rdf:ID="hasip_destination"/>
34    </owl:onProperty>
35 </owl:Restriction>
```

Um atributo também pode ser definido como booleano e ter valores diferentes para diferentes classes. Por exemplo, a classe **SelfReplicating** possui o atributo ***user_interaction*** para diferenciar as subclasses **Virus** e **Worm** (linha 1 a 5). Para a classe **Virus** o atributo tem valor *false* (linhas 7 a 15), enquanto que para a classe **Worm** tem valor *true* (linhas 17 a 25).

```
1 <owl:DatatypeProperty rdf:about="#user_interaction">
2    <rdfs:domain rdf:resource="#SelfReplicating"/>
3    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
4    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
5 </owl:DatatypeProperty>
6
7 <owl:Class rdf:ID="Virus"/>
8    <owl:Restriction>
9      <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
```

```
10      >false</owl:hasValue>
11      <owl:onProperty>
12        <owl:DatatypeProperty rdf:ID="user_interaction"/>
13      </owl:onProperty>
14    </owl:Restriction>
15 </owl:Class>
16
17 <owl:Class rdf:ID="Worm"/>
18   <owl:Restriction>
19     <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
20     >true</owl:hasValue>
21     <owl:onProperty>
22       <owl:DatatypeProperty rdf:ID="user_interaction"/>
23     </owl:onProperty>
24   </owl:Restriction>
25 </owl:Class>
```

Para os relacionamentos, o construtor owl:ObjectProperty foi utilizado, como mostra o trecho a seguir com os relacionamentos *impliesto_Consequence* (linhas 1 a 4), *actson_Asset* (linhas 6 a 9) e *causes_SecurityIncident* (linhas 11 a 14).

```
1 <owl:ObjectProperty rdf:about="#impliesto_Consequence">
2    <rdfs:range rdf:resource="#Consequence"/>
3    <rdfs:domain rdf:resource="#SecurityIncident"/>
4 </owl:ObjectProperty>
5
6 <owl:ObjectProperty rdf:about="#actson_Asset">
7    <rdfs:range rdf:resource="#Asset"/>
8    <rdfs:domain rdf:resource="#SecurityIncident"/>
9 </owl:ObjectProperty>
10
11 <owl:ObjectProperty rdf:about="#causes_SecurityIncident">
12    <rdfs:range rdf:resource="#SecurityIncident"/>
13    <rdfs:domain rdf:resource="#Attack"/>
14 </owl:ObjectProperty>
```

Os relacionamentos não-hierárquicos podem ter restrições de cardinalidade, como mostra o trecho de código a seguir. Por exemplo, os relacionamentos *impliesto_Consequence* e *happenson_Asset* têm o mínimo de cardinalidade igual a um (1), o que significa que um incidente de segurança pode, respectivamente, causar mais de uma conseqüência e atingir mais de um ativo (linhas 1 a 16). O relacionamento *causes_SecurityIncident* também tem o mínimo de cardinalidade igual a um (1), indicando que um ataque pode causar pelo menos um incidente de segurança (linhas 17 a 23).

```
1 <owl:Restriction>
2   <owl:onProperty>
3     <owl:ObjectProperty rdf:ID="impliesto_Consequence"/>
4   </owl:onProperty>
```

```
5   <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
6      >1</owl:minCardinality>
7 </owl:Restriction>
8
9  <owl:Restriction>
10   <owl:onProperty>
11     <owl:ObjectProperty rdf:ID="happenson_Asset"/>
12   </owl:onProperty>
13   <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
14     >1</owl:minCardinality>
15  </owl:Restriction>
16
17 <owl:Restriction>
18   <owl:onProperty>
19     <owl:ObjectProperty rdf:ID="causes_SecurityIncident"/>
20   </owl:onProperty>
21   <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
22     >1</owl:minCardinality>
23 </owl:Restriction>
```

4.2.5 Integração

A OntoSec importou alguns conceitos e relacionamentos da Ontologia de Vulnerabilidades desenvolvida por Brandão (2004), uma vez que incidentes de segurança estão freqüentemente relacionados com vulnerabilidades.

A Ontologia de Vulnerabilidades foi importada pela OntoSec utilizando a ferramenta Protégé. Na Protégé, na aba ***Metadata***, é possível importar uma ontologia definindo três campos:

- *Imported URI*: Identifica uma URI da qual uma ontologia é importada da Web. Esse campo é obrigatório. No caso da OntoSec a URI definida foi `http://a.com/OntoVul.owl`.

- *Local File*: Identifica um arquivo de onde importar a ontologia. Caso esse campo seja omitido, a ontologia é importada diretamente da Web. A OntoSec importou a Ontologia de Vulnerabilidades do arquivo `OntoVul.owl`.

- *Prefix for default namespace*: Identifica o rótulo a ser utilizado para representar o espaço de nomes da ontologia importada. O rótulo `ontovul` foi definido para representar a Ontologia de Vulnerabilidades.

A OntoSec não importou toda a Ontologia de Vulnerabilidades. Os seguintes conceitos não foram importados:

- **Product**, com suas subclasses **Hardware** e **Software**, pois ambas as subclasses já estavam na OntoSec como subclasses da classe **Asset**.
- **Worm**, pois também já estava representado como subclasse da classe **Tool**.
- **Consequence**, com suas subclasses **Severity**, **LossofFeatures** e **SystemAffected**, pois elas já estavam representadas na OntoSec. A classe **Severity** está como atributo da classe **SecurityIncident**, a classe **LossofFeatures** está representada pelas subclasses da classe **Consequence**, e a classe **SystemAffected** está representada pelas subclasses da classe **Asset**.

Assim, os conceitos importados foram: **Vulnerability**, **Type**, **Range**, **Correction** e **Supplier**. No entanto, algumas modificações foram feitas após a integração.

Classe Vulnerability. Dois novos relacionamentos foram criados: *happenson_Asset* e *isexploredby_Tool*. Ambos foram criados para substituir os relacionamentos entre essa classe e as classes **Product** e **Worm**, respectivamente.

Classe Type. Na ontologia original essa classe possui os tipos das vulnerabilidades representadas como instâncias diretas da classe, ou seja, sem um atributo relacionado a ela. Após a integração, um atributo, o ***has_vultype***, foi criado para representar esses tipos. Os tipos *SQLInjection*, *BufferOverflow* e *CrossSiteScripting* não foram importados da ontologia original, pois já existiam como tipos de incidentes. Representar um conceito como atributo ao invés de instância direta facilita a manutenção da ontologia, pois deixa claro quais são as instâncias que uma classe pode ter.

Classe Range. Assim como na classe **Type**, o atributo ***has_range*** foi criado para representar as instâncias da classe. Com exceção das instâncias *Local* e *Remote*, que já estavam na OntoSec, todas as instâncias originais foram importadas.

Classe Supplier. O relacionamento original da classe, o *sellsProduct*, foi mantido. No entanto, as classes com as quais ele se relaciona mudaram. Uma vez que a classe **Product** não foi importada, na OntoSec essa classe se relaciona com as subclasses **Hardware** e **Software** diretamente.

O esforço para importar a Ontologia de Vulnerabilidades foi minimizado pelo fato de que a ontologia já era bastante conhecida. Assim, não houve a necessidade de estudá-la antes da integração.

4.2.6 Implementação

A ferramenta Protégé permite que código Java seja gerado com as classes, os relacionamentos e os atributos definidos a partir de uma ontologia, facilitando o desenvolvimento de aplicações. Assim, utilizando o código Java gerado a partir da OntoSec, um sistema de gerenciamento de incidentes de segurança foi modelado e está sendo desenvolvido. Esse sistema tem como principais objetivos: (i) integrar as aplicações desenvolvidas para validar a ontologia, permitindo, primeiramente, que os administradores de segurança possam inserir e consultar informações sobre incidentes de segurança, e (ii) apoiar as decisões gerenciais de segurança tomadas pelos administradores. O Capítulo 6 descreve a modelagem desse sistema.

4.2.7 Manutenção

Durante o seu desenvolvimento, a OntoSec sofreu modificações com relação às especializações dos principais conceitos e às restrições das propriedades , principalmente devido ao amadurecimento adquirido ao longo do trabalho. No entanto, como os principais conceitos foram definidos inicialmente (seguindo a abordagem *middle-out*), o *"core"* da ontologia não sofreu alterações. No entanto, como o domínio de incidentes de segurança é bastante dinâmico, manutenções, principalmente evolutivas, deverão ser realizadas na OntoSec.

4.2.8 Documentação

Os documentos gerados durante o desenvolvimento da OntoSec são os seguintes:

- Diagrama de conceitos e relacionamentos não-hierárquicos, ilustrado na Figura 4.1.
- Hierarquia de Classes da OntoSec, ilustrada na Figura 4.2.
- Vocabulário de Conceitos e de Relacionamentos, apresentado no Apêndice A.
- Código OWL da ontologia.
- Código da aplicação de consulta desenvolvida para validar a OntoSec.
- Modelo conceitual (Figura 6.1) e arquitetura do sistema de gerenciamento de incidentes de segurança, apresentados no Capítulo 6.

4.2.9 Avaliação

A atividade de avaliação da OntoSec está descrita com detalhes nos Capítulos 5 e 6, nos quais são apresentadas a validação da ontologia utilizando alertas de segurança de uma ferramenta de detecção de intrusão, e a modelagem de um sistema de gerenciamento de incidentes de segurança que está sendo desenvolvido com base na ontologia, respectivamente.

4.3 Considerações Finais

Neste capítulo foram descritas as atividades realizadas durante o desenvolvimento da OntoSec. As metodologias `Methontology` e de Noy e McGuinness foram utilizadas para guiar esse desenvolvimento. Para modelar e formalizar a OntoSec a ferramenta Protégé e a linguagem OWL foram utilizadas.

Desenvolver uma ontologia não é uma tarefa trivial, principalmente em um domínio de conhecimento bastante dinâmico como o de incidentes de segurança. A tarefa mais difícil no desenvolvimento da OntoSec foi definir quais e como os conceitos seriam representados e relacionados entre si, ou seja, a fase de Conceituação. No entanto, definir primeiramente os conceitos principais (ou o *core*) e a partir deles estender a ontologia (abordagem *middle-out*), tornou a tarefa menos onerosa. O fato da ontologia ser "centrada" no conceito do "incidente de segurança" também facilitou, pois os outros conceitos e os relacionamentos foram incorporados a partir do incidente.

O uso das metodologias para guiar o desenvolvimento da OntoSec também tornou o trabalho menos oneroso, pois as atividades executadas em cada uma das etapas foram bem definidas. No entanto, a parte inicial do desenvolvimento, principalmente a Conceituação, é bastante empírica e depende muito do *expertise* do projetista.

O próximo capítulo apresenta com detalhes o processo de avaliação da OntoSec, mostrando como a ontologia foi validada por meio de uma aplicação de consultas e como ela foi avaliada utilizando uma máquina de inferências.

Capítulo

5

O Processo de Avaliação da OntoSec

5.1 Considerações Iniciais

A avaliação de ontologias ainda é uma área de pesquisa incipiente. Muitos autores discutem o assunto (Brewster et al., 2004; Corcho et al., 2004; Fernández et al., 1997; Gómez-Pérez, 1999; Gómez-Pérez et al., 1996; Gruninger e Fox, 1995; Uschold, 1996; Uschold e Gruninger, 1996; Uschold e King, 1995), mas não existem modelos formais e padronizados. Segundo Gómez-Pérez (1999), diversos métodos, técnicas e ferramentas foram desenvolvidos para avaliar Sistemas Baseados em Conhecimento (SBC), nos quais a base de conhecimento é formalizada por meio de regras. No entanto, como uma ontologia é formalizada por meio de conceitos organizados em taxonomias, instâncias, relacionamentos e axiomas utilizando linguagens como OWL, por exemplo, outros métodos devem ser utilizados. Mas, independentemente dos métodos e técnicas de avaliação utilizadas, é importante ter em mente que boas ontologias são aquelas que servem ao propósito para o qual elas foram desenvolvidas.

Brank et al. (2005) apresentam um *survey* com algumas abordagens utilizadas para avaliar ontologias. As principais abordagens são:

1. Maedche e Staab (2002) propõem comparar a ontologia com um "padrão" (*golden standard*) existente do domínio, que também pode ser uma ontologia.

2. Brewster et al. (2004) e Chintan et al. (2004) propõem comparar a ontologia com um conjunto de dados ou documentos sobre o domínio de conhecimento, verifi-

cando o quanto do domínio a ontologia representa. Essa abordagem é chamada de *Data-driven*.

3. Porzel e Malaka (2004) propõem a utilização da ontologia em uma aplicação relacionada ao domínio de conhecimento. Essa abordagem é chamada de *Application-based*.

4. Lozano-Tello e Gómez-Pérez (2004) e Sirin et al. (2006) propõem avaliar o quanto a ontologia satisfaz um conjunto de critérios pré-definidos.

Para avaliar a OntoSec, as abordagens 2, 3 e 4 foram utilizadas. A abordagem definida por Maedche e Staab não foi utilizada porque não existe um padrão no domínio de segurança com o qual a ontologia possa ser comparada.

Para aplicar as abordagens selecionadas, um conjunto de alertas de segurança foi utilizado para avaliar o quanto do domínio de incidentes a OntoSec representa, a avaliação com alguns critérios pré-definidos foi realizada, e um sistema de gerenciamento de incidentes de segurança foi modelado. Neste capítulo são apresentadas as atividades realizadas para avaliar a OntoSec utilizando as abordagens 2 e 4, enquanto que o sistema de gerenciamento está descrito no Capítulo 6.

5.2 Avaliação da OntoSec

Para validar a OntoSec, as abordagens de Brewster et al. (2004) e Chintan et al. (2004) foram utilizadas, enquanto que os critérios propostos por Lozano-Tello e Gómez-Pérez (2004) e Sirin et al. (2006) foram utilizados para avaliar (*assessment*) a ontologia. Essas abordagens foram divididas em duas tarefas: Validação e *Assessment*[1].

- **Validação**: Visando a avaliar se a ontologia foi desenvolvida corretamente, e que os conceitos e relacionamentos estão de acordo com os requisitos estabelecidos e com as questões de competência definidas, ou seja, determinar se a ontologia está de acordo com um modelo real do domínio de conhecimento. A validação permite determinar o que a ontologia representa e/ou infere de correto, o que não representa e/ou não infere, ou o que representa e/ou infere de incorreto. Nessa fase, um conjunto de alertas de segurança da ferramenta **Snort** foi utilizado.

- ***Assessment***: Visando a avaliar a ontologia de acordo com os seguintes critérios:

[1]Em português, o termo *assessment* pode ser traduzido para "avaliação". No entanto, para não confundi-lo com a avaliação da ontologia, o termo em inglês é utilizado.

- **Capacidade de expansão**: Refere-se ao esforço de manutenção da ontologia, na qual novos conceitos e propriedades podem ser incorporados sem alterar os conceitos e propriedades já definidos.
- **Classificação**: Refere-se a verificar a hierarquia de classes da ontologia.
- **Consistência**: Uma ontologia está consistente quando ela não possui fatos contraditórios, ou seja, todo fato representado e inferido pela ontologia está de acordo com o mundo real. Na terminologia da lógica de descrição refere-se a verificar a consistência entre uma assertiva *ABox* com respeito a um axioma *TBox* (veja Seção 3.3.2).
- **Expressividade**: Esse critério está relacionado à capacidade de expressão da linguagem utilizada para formalizar a ontologia. (Veja Seção 3.2.3 sobre a linguagem *OWL*).
- **Precisão**: Uma ontologia é precisa (ou concisa) se ela não representa nenhum conceito ou propriedade desnecessária ou sem utilidade.
- **Satisfação**: Checa se é possível que uma classe não tenha instâncias, assim, se uma classe não pode ser satisfeita, definir uma instância para ela torna a ontologia inconsistente.
- **Sensibilidade**: Refere-se a avaliar o quanto pequenas modificações na ontologia podem alterar as propriedades já definidas.
- **Usabilidade**: Refere-se à facilidade com que pessoas ou programas podem utilizar e entender a ontologia.
- **Utilidade**: Uma ontologia é útil se ela pode ser utilizada por pessoas ou programas para aquisição de conhecimento sobre um domínio.

5.2.1 Validação da OntoSec

A validação da OntoSec consistiu em utilizar um conjunto de alertas gerado pela ferramenta **Snort** para avaliar o quanto a ontologia representa do domínio de incidentes (Martimiano e Moreira, 2006b). Essa tarefa foi realizada em duas fases:

- **Fase 1**: Mapeamento dos alertas para a OntoSec[2].
- **Fase 2**: Desenvolvimento de uma aplicação de consultas às informações mapeadas para a OntoSec.

[2]O mapeamento foi implementado pelo aluno Matheus Lorenzo dos Santos em seu trabalho de Iniciação Científica (Santos, 2006), que foi realizado com apoio do CNPq (bolsa PIBIC).

A Figura 5.1 mostra as tarefas realizadas durante a validação.

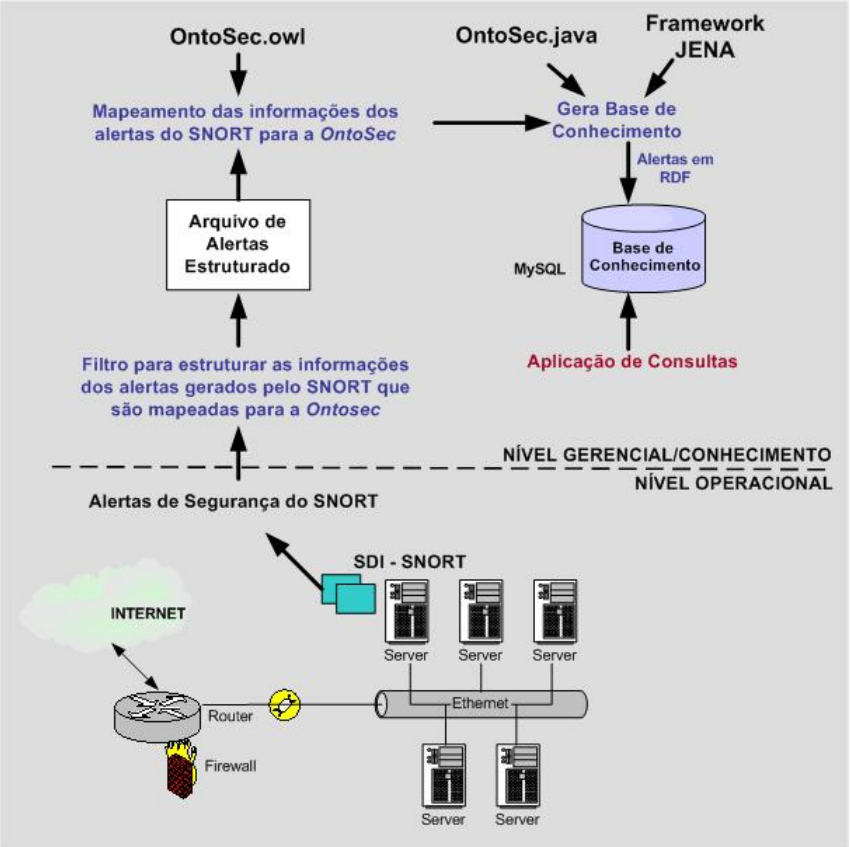

Figura 5.1: Processo de validação da OntoSec.

Fase 1 - O Mapeamento dos Alertas para a OntoSec

Para realizar o mapeamento, o código Java (*OntoSec.java*) gerado pela Protégé a partir da ontologia (*OntoSec.owl*) foi utilizado. Esse código é um "esqueleto" das classes e propriedades da ontologia.

A aplicação de mapeamento foi implementada utilizando o conceito de modularidade. São basicamente três módulos: (i) módulo para leitura e estruturação dos alertas, (ii) módulo para mapear as informações dos alertas para a OntoSec, e (iii) módulo para armazenar os alertas em formato RDF no banco de dados. Essa modularização facilita que informações de alertas de outras ferramentas de gerenciamento de segurança, como *firewalls* por exemplo, possam ser mapeadas para a ontologia, ou seja, facilita a interoperabilidade entre ferramentas de gerenciamento de segurança. Esses módulos são descritos a seguir.

1. **Mapeamento dos tipos de alertas para os tipos de incidentes.**
   Desenvolvimento de um programa que gera uma lista contendo os tipos de alertas (ou classificação dos alertas) que o **Snort** detecta, e realiza o mapeamento desses tipos para os tipos de incidentes definidos na OntoSec. Inicialmente, o mapeamento foi feito de maneira manual, por meio da análise dos tipos de alertas do **Snort** e duas descrições. Para o conjunto de alertas utilizado, o resultado dessa tarefa é apresentado na Tabela 5.1.

Tabela 5.1: Mapeamento dos tipos de ataques do **Snort** para os tipos de incidentes da OntoSec.

Tipos de Alertas do Snort	**Tipos de Incidentes da OntoSec**
Attempted Information Leak	*Scanning*
Attempted User Privilege Gain	*Account Compromise*
Attempted Administrator Privilege Gain	*Root Compromise*
Attempt to login by a default username and password	*Account Compromise, Root Compromise*
Attempted Denial of Service	*Denial of Service*
A suspicious filename was detected	*Malicious Code*
An attempted login using a suspicious username was detected	*Account Compromise*
A Network Trojan was detected	*Malicious Code*
A suspicious string was detected	*Malicious Code*
Executable code was detected	*Malicious Code*
Decode of a RPC query	*Denial of Service*
Detection of a Network Scan	*Scanning*
Misc Attack	*Miscellaneous Attack*
Misc Activity	*Miscellaneous Attack*
Potential Corporate Privacy Violation	*Account Compromise*
Potentially Bad Traffic	*Scanning*
Unknown Traffic	*Scanning*
SCORE! Get the lotion!	*Pornography*
Successful User Privilege Gain	*Account Compromise*
Unsuccessful User Privilege Gain	*Account Compromise*
Successful Administrator Privilege Gain	*Root Compromise*
Web Application Attack	*Cross-Site Scripting*

2. **Mapeamento das conseqüências de um alerta.** Apesar do **Snort** não armazenar essa informação, as conseqüências de um tipo de alerta foram mapeadas de acordo com os tipos de incidentes da OntoSec. Essa tarefa foi realizada primeiramente de maneira manual a partir do mapeamento apresentado na Tabela 5.1, e depois implementada. A Tabela 5.2 mostra o resultado dessa tarefa.

Como mostra a tabela, alguns tipos de incidentes podem ter mais de uma conseqüência, assim como foi definido na ontologia. No entanto, na ocorrência de um incidente de segurança, não necessariamente todas as conseqüências irão acontecer.

Tabela 5.2: Mapeamento dos tipos de incidentes da OntoSec e suas Conseqüências.

Tipos de Incidentes	**Conseqüências**
Account Compromise	*Loss of Privacy, Loss of Confidentiality*
Buffer Overflow	*Loss of Availability*
Cookie Poisoning	*Loss of Confidentiality, Loss of Privacy*
Cross-Site Scripting	*Loss of Availability, Loss of Confidentiality*
Deface	*Loss of Privacy, Loss of Integrity, Loss of Confidentiality*
DoS/DDoS	*Loss of Availability*
Harvesting	*Loss of Confidentiality, Loss of Integrity*
Hijack	*Loss of Confidentiality*
Malicious Code	*Loss of Confidentiality, Remote Execution*
Man in the Middle	*Loss of Confidentiality, Loss of Integrity*
Miscellaneous Attack	*Loss of Confidentiality, Loss of Availability, Loss of Integrity*
Misuse	*Loss of Confidentiality, Loss of Availability, Loss of Integrity*
Natural Disaster	*Natural Damage*
Open Proxy	*Loss of Availability*
Open Relay	*Loss of Availability*
Physical Problem	*Physical Damage*
Probe	*Loss of Availability*
Root Compromise	*Configuration Change, Loss of Privacy*
Scam Phishing	*Loss of Confidentiality, Loss of Integrity*
Scanning	*Loss of Confidentiality, Loss of Integrity*
Social Engineering	*Loss of Confidentiality, Loss of Privacy*
Sniffing	*Loss of Confidentiality, Loss of Integrity*
Spam	*Loss of Availability, Loss of Integrity*
Spoofing	*Loss of Confidentiality, Loss of Integrity*
SQL Injection	*Loss of Confidentiality, Loss of Integrity, Remote Execution*

3. **Mapeamento dos alertas para a OntoSec e geração da base.** Desenvolvimento de um programa que realiza a leitura dos alertas, salvando, em uma estrutura de dados, as principais informações sobre esses alertas. Um subconjunto de alertas gerados pelo **Snort** de 592 KBytes foi utilizado para gerar uma base de conhecimento em RDF de 20 MBytes. Essa base de conhecimento foi utilizada na aplicação de consultas. O exemplo a seguir mostra um alerta do **Snort**, bem como a estrutura de dados gerada para esse alerta[3].

[3]Os números IPs foram suprimidos para preservar a identidade das máquinas envolvidas no alerta.

```
ALERTA
[**] [1:1917:6] SCAN UPnP service discover attempt [**]
[Classification: Detection of a Network Scan] [Priority: 3]
01/20-14:59:17.809869 XXX.XXX.XXX.206:8008 -> XXX.XXX.XXX.250:1900
UDP TTL:4 TOS:0x0 ID:47589 IpLen:20 DgmLen:129
Len: 101

ESTRUTURA GERADA
Descrição: [1:1917:6] SCAN UPnP service discover attempt
Classificação: Detection of a Network Scan
Data: 01/20
Hora: 14:59:17.809869
IP Origem: XXX.XXX.XXX.206 Porta: 8008
IP Destino: XXX.XXX.XXX.250 Porta: 1900
Protocolo: UDP
Prioridade: 3
```

A Tabela 5.3 mostra para quais atributos e classes da OntoSec as informações dos alertas foram mapeadas. A maior parte das informações foram mapeadas diretamente para a classe **SecurityIncident**. Outras informações, como "porta origem" e "porta destino", e "protocolo", foram mapeadas por meio dos relacionamentos entre a classe **SecurityIncident** e as classes **Port** (*uses_Port*) e **Prococol** (*uses_Protocol*), respectivamente. A OntoSec possui um atributo que indica qual o dia da semana que o incidente ocorreu. No entanto, o **Snort** não armazena essa informação. Assim, a partir da data do alerta, o programa de mapeamento calcula o dia da semana automaticamente armazenando-o no atributo ***has_weekday***. A prioridade de um alerta (*Priority*) indica a sua gravidade, podendo ser *Low* (3), *Medium* (2) ou *High* (1).

Tabela 5.3: Mapeamento dos atributos dos alertas do **Snort** para os atributos da OntoSec.

Atributos dos Alertas do Snort	**Atributos mapeados**	**Classe**
Descrição	***has_description***	**SecurityIncident**
Classificação	***hassecurity_incident_type***	**SecurityIncident**
Data	***has_date***	**SecurityIncident**
Hora	***has_time***	**SecurityIncident**
IP Origem	***hasip_source***	**SecurityIncident**
Porta Origem	***has_sourceportnumber***	**Port**
IP Destino	***hasip_destination***	**SecurityIncident**
Porta Destino	***has_destinyportnumber***	**Port**
Prioridade	***has_severity***	**SecurityIncident**
Protocolo	***hasprotocol_name***	**Protocol**
Referência	***has_reference***	**SecurityIncident**

Realizado o mapeamento, os alertas são armazenados em formato RDF, como mostra o exemplo a seguir, criando uma base de conhecimento sobre incidentes. Essa base de conhecimento é armazenada em um banco de dados MySQL com o auxílio do *framework* Jena. Para cada objeto RDF, uma identificação é criada. Para o exemplo apresentado, a identificação para o incidente é `rdf:nodeID=‘‘A6984’’`. As identificações `rdf:nodeID=‘‘A1795’’`, `rdf:nodeID=‘‘A5234’’` e `rdf:nodeID=‘‘A6985’’` indicam com quais objetos o incidente possui os relacionamentos *impliesto_Consequence*, textituses_Protocol e *uses_Port*, respectivamente.

```
<rdf:RDF
    xmlns:j.0="http://a.com/SecurityIncidentOntology#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <rdf:Description rdf:nodeID="A6984">
      <rdf:type rdf:resource="http://a.com/SecurityIncidentOntology#SecurityIncident"/>
     <j.0:has_description>SCAN UPnP service discover attempt</j.0:has_description>
     <j.0:hassecurity_incident_type>Scanning</j.0:hassecurity_incident_type>
     <j.0:has_date>2006-01-20</j.0:has_date>
     <j.0:has_time>14:59:17</j.0:has_time>
     <j.0:hasip_source>XXX.XXX.XXX.206</j.0:hasip_source>
     <j.0:hasip_destination>XXX.XXX.XXX.250</j.0:hasip_destination>
     <j.0:has_weekday>Friday</j.0:has_weekday>
      <j.0:has_severity>Low</j.0:has_severity>
      <j.0:impliesto_Consequence rdf:nodeID="A1795"/>
     <j.0:uses_Protocol rdf:nodeID="A5234"/>
     <j.0:uses_Port rdf:nodeID="A6985"/>
    </rdf:Description>

    <rdf:Description rdf:nodeID="A1795">
    <rdf:type rdf:resource="http://a.com/SecurityIncidentOntology#Consequence"/>
       <j.0:has_consequence>Loss of Integrity</j.0:has_consequence>
       <j.0:has_consequence>Loss of Confidentiality</j.0:has_consequence>
    </rdf:Description>

   <rdf:Description rdf:nodeID="A5234">
     <rdf:type rdf:resource="http://a.com/SecurityIncidentOntology#Protocol"/>
     <j.0:hasprotocol_name>UDP</j.0:hasprotocol_name>
    </rdf:Description>

    <rdf:Description rdf:nodeID="A6985">
     <rdf:type rdf:resource="http://a.com/SecurityIncidentOntology#Port"/>
   <j.0:has_sourceportnumber>8008/j.0:has_sourceportnumber>
     <j.0:has_destinyportnumber>1900</j.0:has_destinyportnumber>
    </rdf:Description>
</rdf>
```

Optou-se por utilizar um banco de dados para armazenar a base de conhecimento pelo fato de que as consultas são realizadas de maneira mais eficiente e otimizada. Além disso,

o banco de dados é persistente e o *framework* Jena facilita a integração entre a aplicação e o banco. No momento do armazenamento no banco de dados, o Jena cria toda a estrutura de tabelas utilizando grafos RDF. A Figura 5.2 mostra como é o grafo RDF equivalente ao exemplo apresentado.

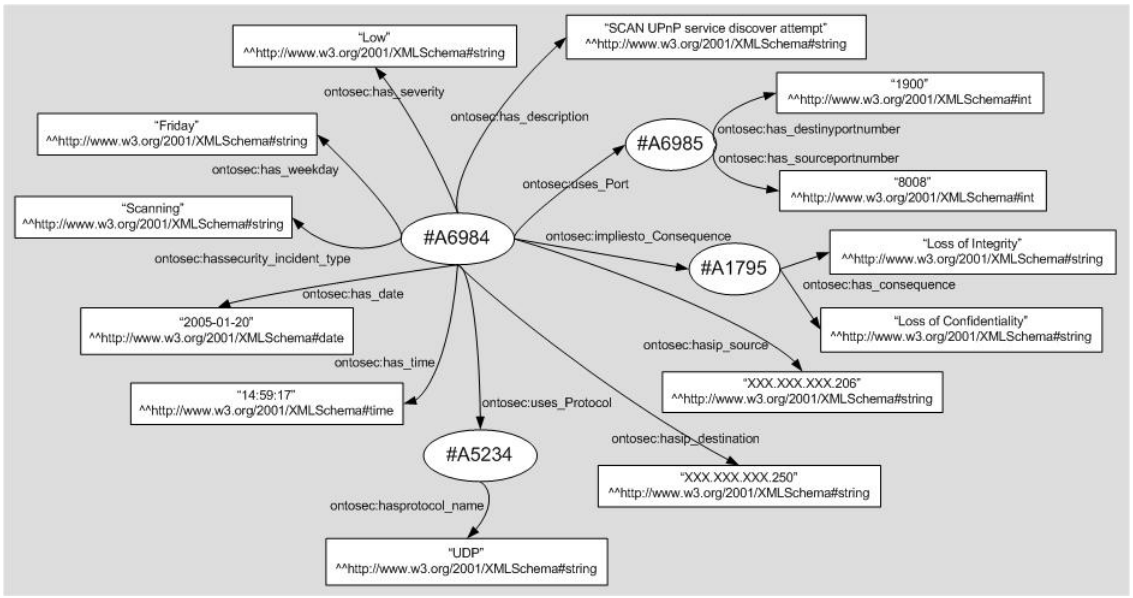

Figura 5.2: Exemplo de um incidente armazenado em formato RDF.

Nem todas as classes e propriedades da OntoSec foram instanciados a partir do mapeamento dos alertas do **Snort** para a OntoSec, pois a ontologia representa mais informações do que os alertas, tais como ativos alvos do incidente e ferramentas utilizadas. Assim, para instanciar mais classes e propriedades da OntoSec, um sistema de gerenciamento de incidentes de segurança foi modelado e está em desenvolvimento.

Fase 2 - A Aplicação de Consultas

Uma vez criada a base de conhecimento, consultas podem ser feitas à ontologia. Assim, uma aplicação de consultas foi implementada utilizando a linguagem Java e o *framework* Jena. E para realizar as consultas, a linguagem SPARQL foi utilizada.

Algumas consultas foram criadas para validar a ontologia. Essas consultas estão baseadas nas "questões de competências" definidas na fase de Aquisição de Conhecimento e foram criadas levando em consideração as informações que foram possíveis de ser mapeadas dos alertas do **Snort** para a OntoSec. A seguir, algumas das consultas que foram pré-definidas na aplicação utilizando a linguagem SPARQL são apresentadas.

As consultas 1 e 2 possibilitam que os administradores possam saber quais incidentes ocorreram em um determinado dia e quais incidentes foram causados por códigos maliciosos, respectivamente. No caso da consulta 2, se muitos incidentes estão sendo causados por códigos maliciosos (*Malicious Code*), como vírus, *worms* ou *trojan horses*, os administradores podem realizar atualizações mais freqüentes nos programas, e podem conscientizar os usuários de maneira mais eficiente. Além disso, combinando a consulta 2 com a consulta 1, é possível identificar em quais datas incidentes causados por códigos maliciosos são mais freqüentes. Com essa informação, os administradores podem se prevenir de ataques com códigos maliciosos que agem em determinadas épocas do ano. A Figura 5.3 mostra as respostas para a consulta 1.

```
Questão de Competência 1: Quais incidentes ocorreram no dia 15 de janeiro de 2006?

Consulta 1:
Prefix ab: <http://a.com/SecurityIncidentOntology#>
Select Distinct ?incident ?type
Where { ?x ab:has_date "2006-01-15";
ab:has_description ?incident;
ab:hassecurity_incident_type ?type} Order by ?x
```

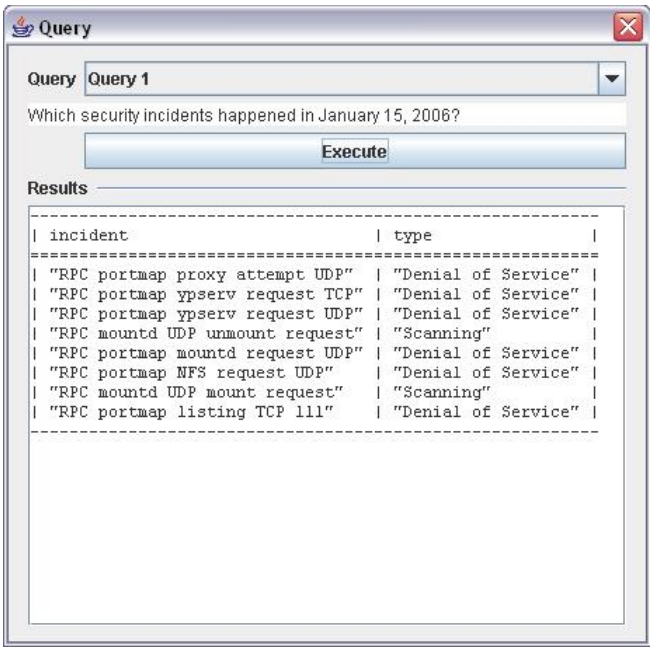

Figura 5.3: Respostas para a consulta 1.

```
Questão de Competência 2: Quais incidentes foram causados por códigos maliciosos?

Consulta 2:
Prefix ab: <http://a.com/SecurityIncidentOntology#>
```

```
Select distinct ?incident ?date
Where { ?a ab:hassecurity_incident_type "Malicious Code";
ab:has_description ?incident;
ab:has_date ?date}
```

A consulta 3 possibilita que os administradores identifiquem incidentes que tiveram como conseqüência uma execução remota (*Remote Execution*). A partir desse tipo de consulta, é possível correlacionar tipos de incidentes com conseqüências específicas.

```
Questão de Competência 3: Quais incidentes tiveram como conseqüência uma execução remota?

Consulta 3:
Prefix ab: <http://a.com/SecurityIncidentOntology#>
Select Distinct ?incident ?type ?date
Where { ?x ab:impliesto_Consequence ?a;
ab:has_description ?incident;
ab:hassecurity_incident_type ?type;
ab:has_date ?date.
?a ab:impliesto_Consequence "Remote Execution"} Order by ?x
```

A consulta 4 permite correlacionar a severidade de um incidente com as conseqüências que ele pode causar. Assim, os administradores podem identificar quais são os tipos de incidentes que mais causam problemas graves no sistema.

```
Questão de Competência 4: Quais são os tipos de incidentes mais graves (severidade) que
acontecem freqüentemente no sistema, e quais suas conseqüências? (correlação entre
tipo de incidente, severidade e conseqüência)

Consulta 4:
Prefix ab: <http://a.com/SecurityIncidentOntology#>
Select Distinct ?incident ?type ?consequence
Where { ?x ab:impliesto_Consequence ?a;
ab:has_description ?incident;
ab:hassecurity_incident_type ?type;
ab:has_severity "High".
?a ab:impliesto_Consequence ?consequence} Order by ?x;
```

Também é possível que os administradores saibam quais são os tipos de incidentes de segurança que mais ocorreram no sistema, como mostra o exemplo da consulta 5. Como pode ser visto pelos resultados apresentados na Figura 5.4, o número de incidentes envolvendo negação de serviço (incidente do tipo *Denial of Service*) é bem maior quando comparado com os outros dois tipos de incidentes, 85,67% do total. Com essa informação, os administradores podem melhorar o processo de configuração e atualização dos sistemas para evitar que, por exemplo, vulnerabilidades existam e possam ser exploradas para causar negação de serviço.

Mesmo com um pequeno subconjunto de alertas de segurança mapeados para OntoSec, foi possível demonstrar, por meio das consultas apresentadas, o potencial da

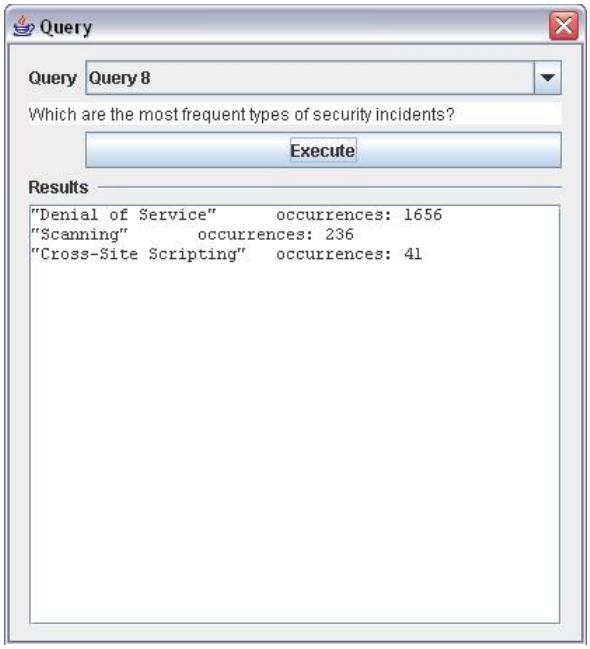

Figura 5.4: Respostas para a consulta 5.

ontologia em fornecer informações sobre problemas de segurança aos administradores, apoiando-os em suas decisões.

5.2.2 Assessment da OntoSec

Os critérios utilizados para avaliar a OntoSec são aqueles apresentados no início deste capítulo. Critérios como consistência, classificação e expressividade são mais objetivos e podem ser avaliados por meio de uma máquina de inferência, por exemplo. No entanto, critérios como capacidade de expansão, satisfação, sensibilidade, *conciseness*, utilidade e usabilidade são subjetivos, o que torna a tarefa de avaliar uma ontologia de acordo com esses critérios mais difícil, pois não se pode medir, por exemplo, a capacidade de expansão da ontologia, ou sua usabilidade. Esses critérios dependem mais da sensibilidade do avaliador.

Para avaliar a OntoSec com os critérios objetivos, a máquina de inferência Pellet (versão 1.3-beta2) foi utilizada. Em seu relatório de resultados, a máquina mostra basicamente as seguintes informações:

- Tempo total para realizar a avaliação.
- Tempo para carregar a ontologia (em memória).
- Tempo para validar a espécie da OWL.

- Tempo para classificar a ontologia.
- Tempo para verificar a consistência da ontologia.
- Se a ontologia é consistente ou não.
- Espécie da OWL utilizada.
- Expressividade da linguagem de formalização.
- Hierarquia de classes da ontologia.

Como mostra a Tabela 5.4, de acordo com a Pellet, a OntoSec é consistente, está formalizada na linguagem *OWL DL* e tem expressividade `SHOIF(D)`. Basicamente, a máquina resume sua avaliação em definir se a ontologia é ou não consistente. Para realizar a avaliação, a máquina considerou as 49 classes e as 94 propriedades da OntoSec, sendo 36 relacionamentos não-hierárquicos e 58 atributos.

Tabela 5.4: Avaliação da OntoSec realizada pela máquina de inferência Pellet.

Espécie OWL	**Expressividade DL**	**Consistente**
DL	`SHOIF(D)`	Sim

A Pellet pode tanto ser executada integrada à Protégé quanto por linha de comando. Para a avaliação da OntoSec, ela foi executada por linha de comando, pois, nesse caso, não existe a sobrecarga da máquina virtual Java quando várias avaliações são realizadas.

Como a Pellet realiza suas tarefas de avaliação em determinado tempo, foram coletadas 30 amostras independentes (em seqüência) para realizar a média dos tempos gastos na avaliação. Um computador Pentium 4 com 1.8GHz de *clock* e 1GByte de memória RAM foi utilizado. Um conjunto de 30 amostras foi coletado para futuramente realizar comparações com outras máquinas de inferência com base em uma avaliação estatística.

A Tabela 5.5 mostra as médias de tempo (em segundos), juntamente com os desvios padrão, gastos para avaliação da ontologia. Na média, a máquina gastou 5,53 segundos para realizar toda a avaliação, com desvio padrão de 0,33 segundos. Para carregar a ontologia e validar a espécie da OWL utilizada, a Pellet gastou na média, respectivamente, 3,47 e 1,60 segundos. E para verificar a consistência e classificar foram gastos 0,02 e 0,44 segundos em média, respectivamente.

Esses números são expressivos considerando que a OntoSec possui 49 classes e 94 propriedades. Para avaliar as ontologias *Financial*[4] e *Food*[5], que possuem número de

[4] `http://www.cs.put.poznan.pl/alawrynowicz/financial.owl`
[5] `http://www.w3.org/2001/sw/WebOnt/guide-src/food.owl`.

classes próximo ao número de classes da OntoSec, 59 e 64 classes, respectivamente, a máquina gastou 1,57 segundos e 7,34 segundos (Sirin et al., 2006)[6].

Tabela 5.5: Média e desvio padrão dos tempos para a máquina de inferência Pellet avaliar a OntoSec.

Métrica	Tempo Total	Carregar	Validar OWL	Consistência	Classificação
Média	5,53	3,47	1,60	0,02	0,44
Desvio Padrão	0,33	0,17	0,17	0,01	0,03

As Figuras 5.5 e 5.6 mostram os valores das 30 amostras coletadas. Para carregar a ontologia, a menor amostra apresentou um tempo de 3,31 segundos e a maior um tempo de 3,98. Para validar a espécie OWL o menor e maior tempo foram, respectivamente, 1,42 e 2,17 segundos. Já para classificar a ontologia, a máquina gastou na menor amostra 0,41 segundos e na maior amostra 0,59 segundos. E para verificar a consistência, foram gastos na menor amostra 0,02 segundos e na maior 0,06 segundos.

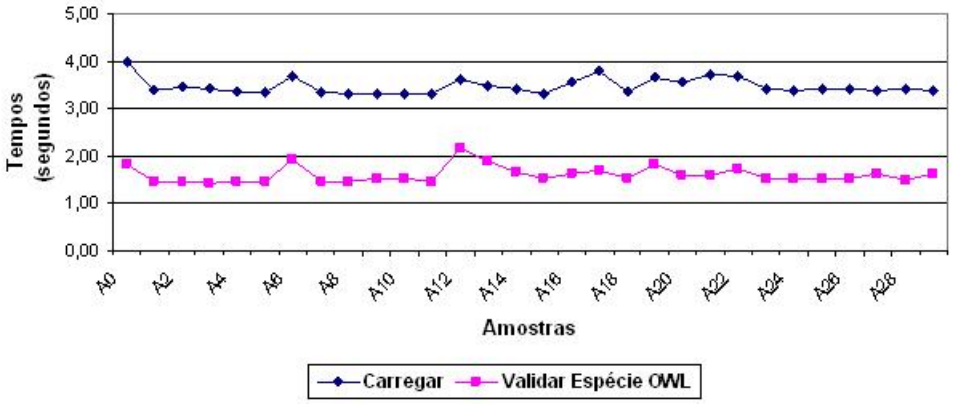

Figura 5.5: Tempos para carregar a OntoSec e validar a espécie OWL.

A seguir, a avaliação da OntoSec de acordo com os critérios subjetivos é apresentada.

Capacidade de Expansão. A partir da atual versão da OntoSec, novos conceitos podem ser incluídos. Seria interessante, por exemplo, incluir na ontologia quais medidas são necessárias para se recuperar de um determinado incidente de segurança, criando uma classe para representar o conceito *Counter Measure*. Essa classe se relacionaria com a classe **SecurityIncident**, indicando que um incidente de segurança *possui* uma contra-medida. Também nesse sentido, seria interessante estabelecer

[6]Média de 10 amostras independentes coletadas em um Pentium Centrino 1.6GHz com 1.5GByte de memória RAM.

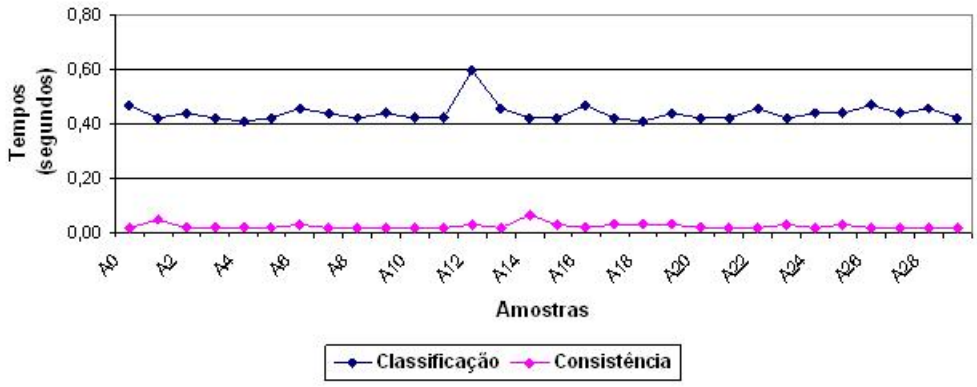

Figura 5.6: Tempos para classificar e verificar a consistência da OntoSec.

responsabilidades sobre a implantação e execução dessas medidas, criando uma classe para representar o conceito *Security Administrator*.

Outro exemplo, seria estender as classes **Worm** e **Virus**. A partir dessas classes, seria possível criar novas "ontologias", definindo, por exemplo, os tipos de *worms* e *virus*, suas abrangências, seus meios de propagação, meios de detecção e prevenção, entre outros. No caso da classe **Asset** que representa os ativos que podem ser alvos de incidentes, seria interessante também estender o conceito *Security Tool*. Na OntoSec os tipos de ferramentas de segurança estão representados como instâncias da classe. No entanto, elas também poderiam estar representadas como subclasses, podendo assim, ser especializadas. Por exemplo, criando uma classe **IntrusionDetectionSystem**, seria possível representar os tipos de sistemas de detecção de intrusão e suas especificidades. Esses exemplos mostram o quanto a OntoSec pode ser utilizada para o desenvolvimento de outras ontologias, contribuindo para que o domínio de segurança possa ser representado por ontologias.

Precisão e Satisfação. Os conceitos representados na ontologia estão de acordo com os conceitos existentes no domínio de incidentes de segurança, tais como: o **agente** executor, os **alvos** do incidente, as **conseqüências** que podem ocorrer, as **ferramentas** que podem ser utilizadas para executar um **ataque** e causar um incidente, e as **vulnerabilidades** que podem ser exploradas. Esses são os conceitos que permitem aos administradores de segurança conhecer melhor seu cenário de incidentes e reagir de forma mais eficiente aos problemas que possam ocorrer, se recuperando desses problemas e, melhor ainda, evitando que eles aconteçam novamente. Portanto, todos

os conceitos são importantes e têm utilidade na ontologia, podendo ser satisfeitos com informações sobre os incidentes.

Sensibilidade. A partir da experiência em desenvolver a OntoSec, pode-se dizer que toda ontologia é sensível a modificações. O que pode ser diferente é o grau de sensibilidade, que pode ser maior ou menor. Se durante o desenvolvimento, o *core* da ontologia é bem definido e sofre poucas modificações ao longo do processo, o grau de sensibilidade é menor. Modificar a cardinalidade de alguma restrição, o tipo de um atributo ou criar um atributo novo para uma classe também não eleva o grau de sensibilidade da ontologia. Por outro lado, se o *core* muda constantemente, e novas classes, atributos e relacionamentos são criados ou removidos, o grau de sensibilidade é elevado. No entanto, se o domínio de conhecimento sofreu uma grande mudança, a ontologia deve ser modificada para representar essa mudança. Nesse caso, o grau de sensibilidade elevado não pode impedir a modificação, pois a ontologia ficará inconsistente com relação ao domínio se a modificação não for realizada.

Usabilidade. Como a ontologia foi desenvolvida a partir do conceito de "incidente de segurança", ela é simples de ser compreendida, pois todos os outros conceitos estão relacionados ao conceito incidente de segurança. Além disso, o uso da linguagem OWL para formalizar a OntoSec facilita o entendimento de suas classes e propriedades.

Utilidade. Tanto administradores de segurança quanto ferramentas de auxílio ao gerenciamento de segurança podem utilizar a OntoSec. Os administradores podem utilizar a ontologia para centralizar em uma única ferramenta informações sobre incidentes de segurança, agregando diferentes fontes de alertas de segurança, como mostra a proposta do sistema de gerenciamento do próximo capítulo. Já as ferramentas de auxílio ao gerenciamento podem utilizar a ontologia como padrão para gerar alertas de segurança.

Além desses potenciais usuários, a OntoSec também pode auxiliar na definição de novas regras da política de segurança de uma organização, pois ela é fonte de conhecimento dos problemas de segurança ocorridos. Por exemplo, se por meio de um sistema de gerenciamento de incidentes baseado na ontologia, os administradores sabem que os usuários estão abusando dos recursos da organização (tipo de incidente *Misuse*), regras mais rígidas podem ser estabelecidas na política de segurança. Se muitos incidentes ocorrem devido a defeitos freqüentes de dispositivos de E/S

(problemas físicos - *Physical Problem*), novas regras para melhorar o processo de manutenção de equipamentos podem ser criadas.

5.3 Considerações Finais

Neste capítulo foi descrito o processo para avaliar a OntoSec. Esse processo consistiu em validar a ontologia com um conjunto de alertas de segurança, desenvolver uma aplicação de consulta para questionar a ontologia, e avaliar a ontologia de acordo com um conjunto de critérios definidos na literatura.

Apesar dos alertas gerados pelo **Snort** não terem instanciado todas as classes e propriedades da OntoSec, o processo de validação mostrou que a ontologia representa informações importantes relacionadas a um incidente de segurança. Mesmo com poucas informações sendo mapeadas para a ontologia, a aplicação de consultas mostrou o potencial da ontologia em responder perguntas sobre os incidentes de segurança, possibilitando que os administradores aprendam com os problemas de segurança de seu ambiente de trabalho e possam tomar decisões de maneira mais eficiente. Na validação, apenas alertas de uma única ferramenta foram utilizados. No entanto, para complementar as informações sobre os incidentes, é necessário mapear alertas de outras ferramentas de segurança, aprimorando, assim, o processo de validação da ontologia. Por exemplo, um antivírus poderia gerar um alerta de código malicioso, e esse alerta poderia estar relacionado a um alerta do tipo *Malicious Code* gerado pelo **Snort**.

O processo de avaliação também teve grande importância, contudo, algumas ponderações devem ser feitas. Critérios como consistência, classificação e expressividade são avaliados de forma mais objetiva, facilitando o processo de avaliação. Esses são critérios que podem ser avaliados por uma máquina de inferência (Pellet, por exemplo) com parâmetros bem definidos por lógicas específicas para descrição de conceitos, como a lógica de descrição. No entanto, critérios como capacidade de expansão, precisão, satisfação, sensibilidade, usabilidade e utilidade são bastante subjetivos, pois dependem muito da experiência do desenvolver da ontologia no domínio de conhecimento.

Apesar da subjetividade, esses critérios podem ser avaliados de maneira mais fácil se alguns cuidados são tomados durante o desenvolvimento da ontologia. Se os objetivos da ontologia, o domínio de conhecimento e os potenciais usuários estão bem claros e definidos nas fases de Planejamento e Especificação, avaliar a precisão, e a utilidade torna-se mais fácil, pois se sabe exatamente o que a ontologia deve representar. Se a ontologia está baseada nos principais conceitos do domínio, ou seja, representa o *core* do domínio, sua capacidade de expansão e sua sensibilidade são mais claras.

O próximo capítulo mostra as principais características propostas para um sistema de gerenciamento de segurança baseado na OntoSec.

Capítulo

6

Gerenciamento de Incidentes de Segurança utilizando a OntoSec

6.1 Considerações Iniciais

Os administradores de segurança devem lidar continuamente com a grande quantidade de informações sobre incidentes que são geradas por diferentes ferramentas em diferentes formatos. Nesse sentido, existe a necessidade de que os administradores possam ter disponível uma ferramenta para unificar essas informações em formato e local únicos, a fim de facilitar seu gerenciamento e tornar mais eficiente a tomada de decisões quando problemas acontecem.

Assim, para apoiar as decisões tomadas pelos administradores para gerenciar segurança e para mostrar como uma ferramenta pode ser implementada a partir de uma ontologia, um sistema de gerenciamento de incidentes de segurança foi modelado e está sendo desenvolvido, e suas principais características são descritas neste capítulo.

6.2 O Sistema de Gerenciamento

O sistema de gerenciamento de incidentes de segurança permitirá unificar informações de diversas ferramentas, tais como sistemas de detecção de intrusão, *firewalls*, antivírus, *scanners*, entre outras, facilitando a interoperabilidade entre essas ferramentas. A partir

dessa unificação, os administradores poderão correlacionar as informações sobre incidentes de maneira mais eficiente[1].

A partir dos principais conceitos e propriedades da OntoSec, um modelo conceitual em UML (*Unified Modeling Language*) (Larman, 2004) foi desenvolvido e está ilustrado na Figura 6.1. Esse modelo define as classes e os relacionamentos da estrutura de dados utilizada pelo sistema de gerenciamento.

O sistema de gerenciamento proposto possui inicialmente os módulos descritos a seguir e apresentados na Figura 6.2.

1. **Módulo de Mapeamento**. Permitirá mapear alertas de segurança de diferentes ferramentas para um padrão único de representação definido pela OntoSec. Esse módulo aproveitará o mapeamento já realizado dos alertas de segurança do **Snort** descrito no Capítulo 5. Para cada ferramenta de segurança, um módulo de mapeamento deve ser implementado. Uma vez mapeados, os alertas são armazenados em formato RDF em um banco de dados com o auxílio do *framework* Jena.

2. **Módulo de Consultas**. Permitirá que consultas sejam realizadas pelos administradores à base de incidentes de segurança. Esse módulo é o mesmo apresentado no Capítulo 5, mas está sendo aprimorado para realizar novas consultas. A princípio, as consultas serão realizadas utilizando a linguagem SPARQL, mas outras linguagens de consulta em RDF poderão ser utilizadas, como RDQL. Esse módulo interage com o Módulo de Inferência.

3. **Módulo de Inferência**. Permitirá inferir informações sobre os incidentes de segurança, utilizando a capacidade de representação da ontologia e uma máquina de inferência (Pellet, por exemplo), e facilitando a correlação automática entre os incidentes.

4. **Módulo de Inserção**. Permitirá que os administradores realizem inserção de incidentes de segurança que não sejam registrados pelas ferramentas automáticas, como por exemplo, incidentes que utilizem engenharia social ou incidentes que sejam causados por eventos da natureza. Além disso, esse módulo também possibilitará que os administradores complementem as informações sobre os incidentes registradas pelas ferramentas de segurança automáticas.

[1] O sistema de gerenciamento está sendo desenvolvido com a ajuda dos alunos Paula Cristina Berengue e Rafael Freitas Planello.

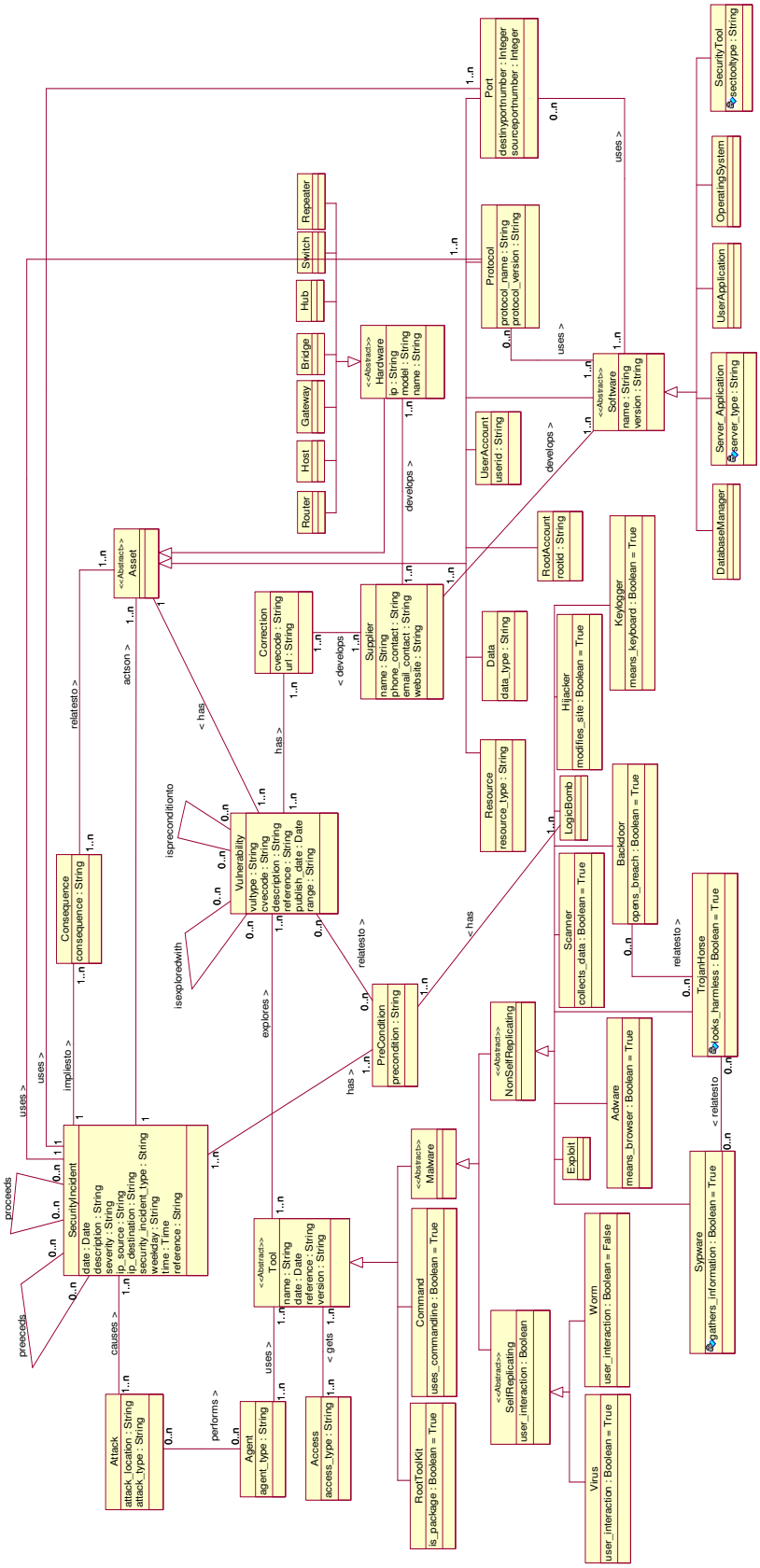

Figura 6.1: Modelo Conceitual do Sistema de Gerenciamento de Incidentes de Segurança.

5. **Módulo de Visualização**. Permitirá aos administradores visualizarem os incidentes já mapeados a partir das ferramentas de segurança, possibilitando complementar as informações. Esse módulo interage com o Módulo de Inserção.

6. **Módulo de Relatórios**. Permitirá que relatórios pré-definidos sejam gerados pelos administradores a fim de obter um cenário diário, semanal e mensal dos incidentes de segurança ocorridos.

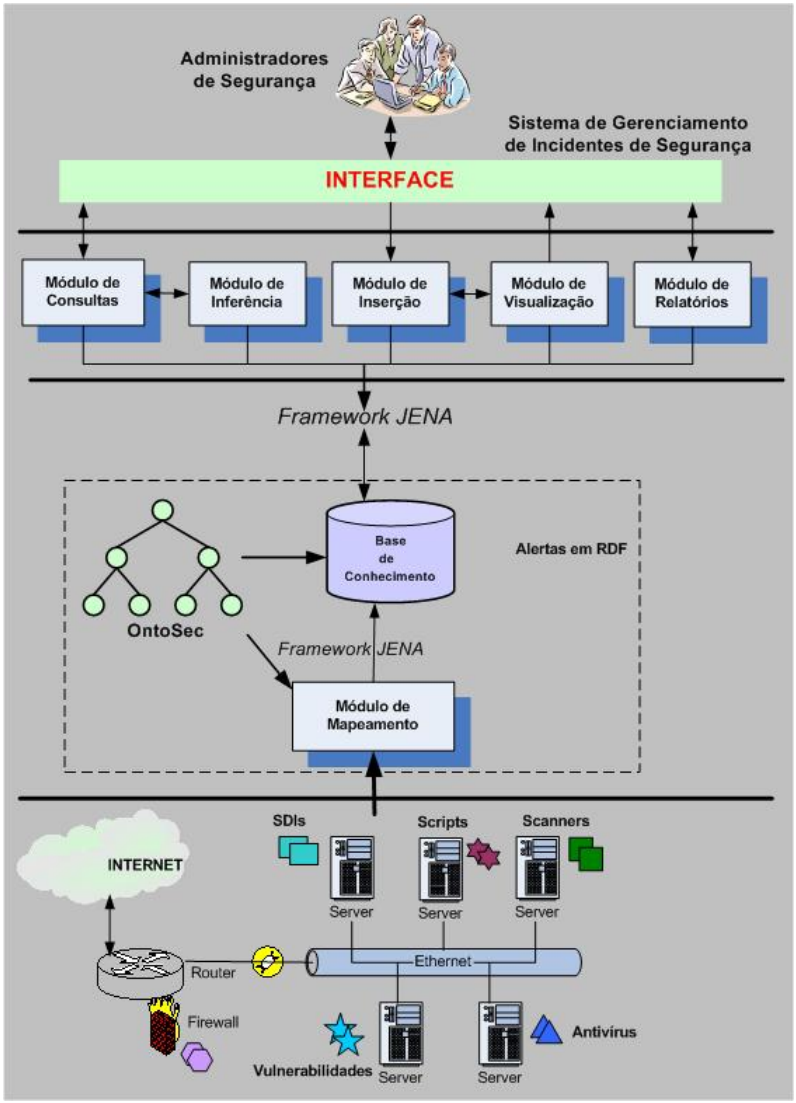

Figura 6.2: Arquitetura do Sistema de Gerenciamento de Incidentes de Segurança.

A seguir, os módulos de inserção e de consultas são descritos com mais detalhes, pois já estão em desenvolvimento.

6.2.1 Módulo de Inserção

Por meio do módulo de inserção, informações sobre incidentes de segurança ocorridos no ambiente de rede de uma organização podem ser inseridas. A seguir, um exemplo de quais informações sobre um incidente de segurança podem ser inseridas no sistema de gerenciamento é apresentado.

O exemplo apresenta um problema reportado pela Microsoft[2] em 2005 envolvendo a exploração de uma vulnerabilidade ("CVE-2005-1983") no serviço de *plug and play* do Windows 2000[3]. A vulnerabilidade podia ser explorada pelo *worm Zobot.A*[4]. Como conseqüência da exploração dessa vulnerabilidade, um usuário comum do sistema poderia obter maiores privilégios no sistema (*Configuration Change*). O Apêndice B mostra as informações divulgadas pelos boletins da Microsoft e do CVE a respeito dessa vulnerabilidade.

Uma vez ocorrido um incidente de segurança devido a esse problema, as informações são armazenadas em RDF pelo módulo de inserção, ou pelo módulo de mapeamento de maneira automática, como apresentado no exemplo a seguir[5].

```
<rdf:RDF
    xmlns:j.0="http://a.com/SecurityIncidentOntology#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >

    <rdf:Description rdf:nodeID="Agent1">
      <rdf:type rdf:resource="http://a.com/SecurityIncidentOntology#Agent"/>
      <j.0:hasagent_type>Human</j.0:hasagent_type>
      <j.0:usesTool rdf:nodeID="Worm1"/>
      <j.0:performsAttack rdf:nodeID="Attack1"/>
    </rdf:Description>

    <rdf:Description rdf:nodeID="Worm1">
      <rdf:type rdf:resource="http://a.com/SecurityIncidentOntology#Worm"/>
      <j.0:hasname_tool>Zobot.A</j.0:hasname_tool>
        <j.0:hasreference_tool>http://www.microsoft.com/brasil/security/advisory.zobot.mspx/
            </j.0:hasreference_tool>
        <j.0:hasdate_tool>2005-10-13</j.0:hasdate_tool>
      <j.0:exploresVulnerability rdf:nodeID="Vul1"/>
    </rdf:Description>

    <rdf:Description rdf:nodeID="Attack1">
      <rdf:type rdf:resource="http://a.com/SecurityIncidentOntology#Attack"/>
      <j.0:hasattack_type>Active</j.0:hasattack_type>
     <j.0:hasattack_location>Local</j.0:hasattack_location>
      <j.0:causes_SecurityIncident rdf:nodeID="SecInc1"/>
```

[2] http://www.microsoft.com/brasil/security/advisory/899588.mspx/. Acesso em 18/07/2006.
[3] http://www.microsoft.com/technet/Security/bulletin/ms05-039.mspx/.
[4] http://www.microsoft.com/brasil/security/advisory.zobot.mspx/.
[5] Nem todas as informações dos boletins foram mapeadas para o exemplo.

```
 </rdf:Description>

 <rdf:Description rdf:nodeID="SecInc1">
   <rdf:type rdf:resource="http://a.com/SecurityIncidentOntology#SecurityIncident"/>
  <j.0:has_description>Explores Plug and Play vulneratility in Windows</j.0:has_description>
  <j.0:hassecurity_incident_type>Malicious Code</j.0:hassecurity_incident_type>
  <j.0:has_date>2005-09-20</j.0:has_date>
  <j.0:has_time>13:59:01</j.0:has_time>
  <j.0:has_reference>http://www.microsoft.com/brasil/security/advisory/899588.mspx/</j.0:has_reference>
  <j.0:hasip_source>192.168.0.5</j.0:hasip_source>
  <j.0:hasip_destination>10.0.0.3</j.0:hasip_destination>
  <j.0:has_weekday>Tuesday</j.0:has_weekday>
   <j.0:has_severity>High</j.0:has_severity>
   <j.0:impliesto_Consequence rdf:nodeID="Conseq1"/>
  <j.0:actson_Asset rdf:nodeID="OS1"/>
 </rdf:Description>

 <rdf:Description rdf:nodeID="Conseq1">
 <rdf:type rdf:resource="http://a.com/SecurityIncidentOntology#Consequence"/>
    <j.0:has_consequence>Configuration Change</j.0:has_consequence>
 </rdf:Description>

<rdf:Description rdf:nodeID="OS1">
  <rdf:type rdf:resource="http://a.com/SecurityIncidentOntology#OperatingSystem"/>
  <j.0:hasname_sw>Windows 2000</j.0:hasname_sw>
  <j.0:hasVulnerability rdf:nodeID="Vul1"/>
  <j.0:hasSupplier rdf:nodeID="Supplier1"/>
 </rdf:Description>

 <rdf:Description rdf:nodeID="Vul1">
  <rdf:type rdf:resource="http://a.com/OntoVul#Vulnerability"/>
<j.0:has_cvecode>CVE-2005-1983</j.0:has_cvecode>
  <j.0:description>Stack-based buffer overflow in the Plug and Play (PnP) service for Microsoft
     Windows 2000</j.0:description>
  <j.0:reference>http://www.microsoft.com/technet/Security/bulletin/ms05-039.mspx/</j.0:reference>
  <j.0:reference>CERT-VN:VU#998653</j.0:reference>
  <j.0:publish_date>2005-08-09</j.0:publish_date>
  <j.0:hasCorrection rdf:nodeID="Correc1"/>
 </rdf:Description>

  <rdf:Description rdf:nodeID="Supplier1">
  <rdf:type rdf:resource="http://a.com/OntoVul#Supplier"/>
<j.0:name>Microsoft Corporation</j.0:name>
  <j.0:website>http://www.microsoft.com/</j.0:website>
  <j.0:developsCorrection rdf:nodeID="Correc1"/>
 </rdf:Description>

 <rdf:Description rdf:nodeID="Correc1">
  <rdf:type rdf:resource="http://a.com/OntoVul#Correction"/>
  <j.0:url>http://www.microsoft.com/brasil/security/boletins/ms05-039.mspx</j.0:url>
 </rdf:Description>
```

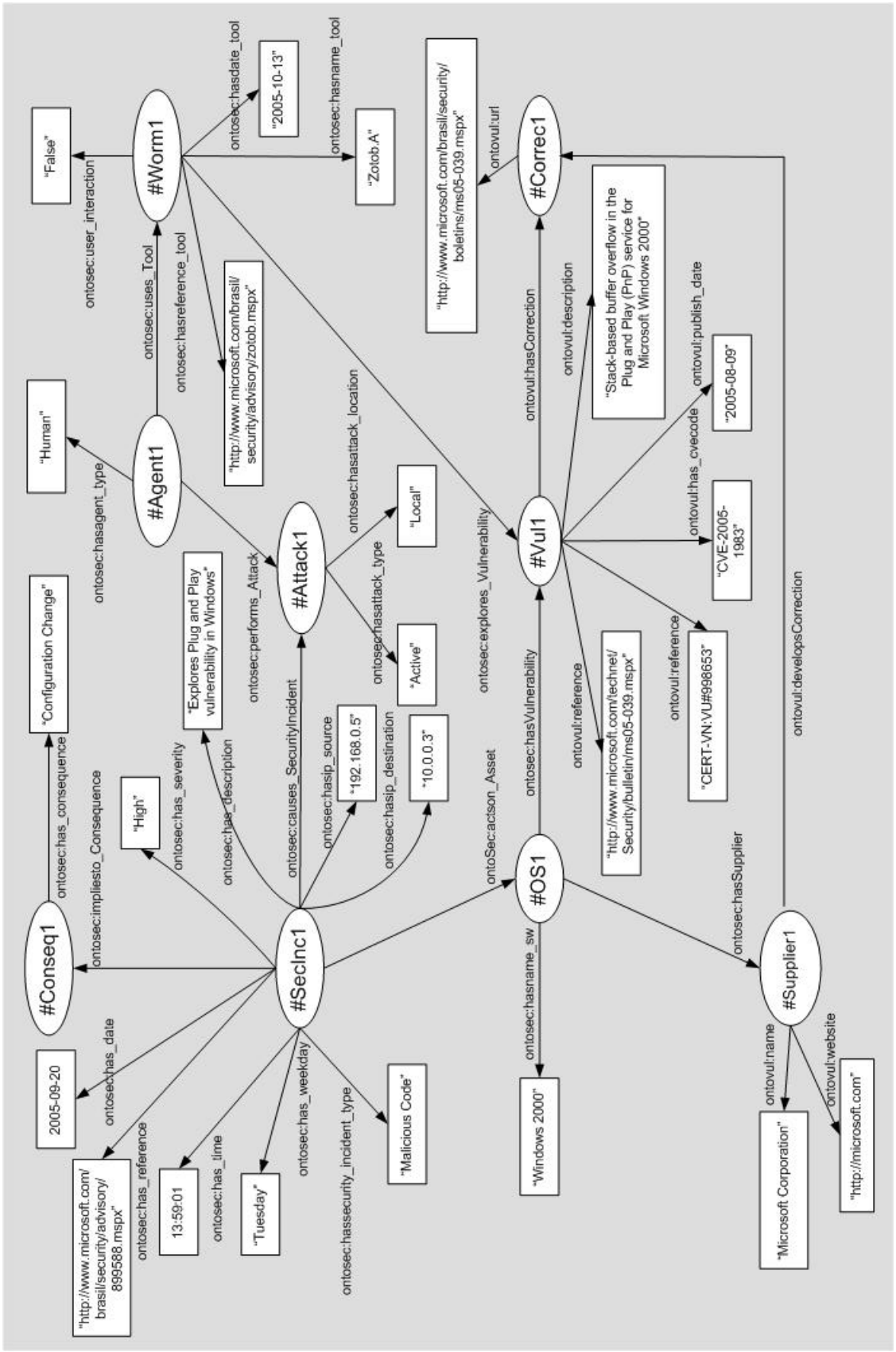

Figura 6.3: Grafo RDF de um incidente causado pelo Worm Zobot.A explorando uma vulnerabilidade do Windows 2000.

A Figura 6.3 mostra o grafo RDF correspondente a esse exemplo[6]. O exemplo mostra o potencial da OntoSec em representar informações sobre incidentes de segurança. Questionando a ontologia, os administradores podem saber quais incidentes foram causados pelo código malicioso (**Malware**) *Worm Zobot.A*. E para responder a essa questão, a hierarquia de classes da ontologia deve ser considerada, uma vez que um *worm* é um código malicioso. Outro exemplo, seria questionar a ontologia para saber quais incidentes foram causados explorando vulnerabilidades no Windows 2000.

A vulnerabilidade apresentada no exemplo também poderia ser explorada no sistema Windows XP SP1. Além de ter como conseqüência uma mudança na configuração do sistema (*Configuration Change*), possibilitando que um usuário comum possa adquirir privilégios no sistema, explorar essa vulnerabilidade também permite a um atacante causar um incidente por meio da execução de código remoto (*Remote Execution*). Com essas informações armazenadas no sistema de gerenciamento, é possível correlacionar vários incidentes por meio de uma mesma vulnerabilidade, como é o caso do exemplo apresentado.

Além de inserir incidentes de segurança, os administradores também poderão complementar informações relacionadas aos alertas mapeados das ferramentas de segurança do ambiente. Por exemplo, a partir de um alerta gerado pelo **Snort**, os administradores podem inserir quais ativos foram atingidos, quais conseqüências e quais ferramentas foram utilizadas.

6.2.2 Módulo de Consultas

O sistema de gerenciamento permitirá que mais informações relacionadas a um incidente de segurança possam ser instanciadas de acordo com a OntoSec, possibilitando inserir incidentes de segurança e realizar novas consultas. Essas consultas podem explorar mais a hierarquia de classes e as propriedades da ontologia. A seguir, três exemplos de consultas a uma base de incidentes de segurança mais rica são apresentados.

A consulta 1 procura correlação entre incidentes de segurança do tipo *Buffer Overflow* e *Denial of Service*. Freqüentemente, uma negação de serviço é precedida por uma sobrecarga em alguma variável do sistema, ou função/procedimento de um aplicativo ou serviço. A consulta poderia ainda ser mais especializada, identificando o ativo que foi alvo de uma sobrecarga, como um servidor Web, ou um servidor de e-mails ou um servidor de arquivos.

[6]No grafo RDF a OntoSec é referenciada pelo rótulo ‘‘`ontosec`’’ ao invés de “`http://a.com/SecurityIncidentOntology`” para facilitar o desenho.

```
Questão de Competência 1: Quais incidentes do tipo ‘‘Buffer Overflow’’ aconteceram antes de um incidente
do tipo ‘‘Denial of Service’’?

Consulta 1:
Prefix ab: <http://a.com/SecurityIncidentOntology#>
Select Distinct ?description ?date
Where { ?x ab:precedes_SecurityIncident ?a;
ab:hassecurity_incident_type "Denial of Service".
?a ab:hassecurity_incident_type "Buffer Overflow";
ab:has_description ?description;
ab:has_date ?date} Order by ?x
```

A partir da consulta 2, os administradores podem identificar as conseqüências de um incidente causado por *spams* que esteja relacionado a um servidor de e-mails, podendo, assim, gerenciar de maneira mais efetiva o servidor e conscientizar os usuários do serviço.

```
Questão de Competência 2: Quais são as conseqüências mais freqüentes causadas por ‘‘spams’’ que
atingem um servidor de e-mails?

Consulta 2:
Prefix ab: <http://a.com/SecurityIncidentOntology#>
Select Distinct ?consequence ?description
Where {?x ab:impliesto_Consequence ?a;
?x ab:actson_Asset ?b.
?a ab:impliesto_Consequence ?consequence;
ab:hasecurity_incident_type "Spam";
ab:has_description ?description.
?b ab:has_servertype "Mail"} Order by ?x
```

Na consulta 3, é possível explorar a propriedade de transitividade que existe na classe **Vulnerability**, na qual uma vulnerabilidade pode ser explorada em conjunto com outra(s) (uso integrado com o Módulo de Inferência). Assim, se uma vulnerabilidade A é explorada em conjunto com uma vulnerabilidade B, e a vulnerabilidade B é explorada em conjunto com a vulnerabilidade C, então a vulnerabilidade A também é explorada em conjunto com a vulnerabilidade C.

O exemplo da consulta 3 foi feito com base no anúncio de atualização do sistema operacional Conectiva Linux divulgado em 22/07/2003[7]. As vulnerabilidades apresentadas pelo anúncio atingem o pacote CUPS (*Common UNIX Printing System*), que é uma solução de impressão, aberta e livre de distribuição para plataforma UNIX. Três vulnerabilidades podem ser exploradas: “CVE-2002-1384”, “CVE-2002-1383” e “CVE-2002-1367”. Se a vulnerabilidade “CVE-2002-1384” for explorada em conjunto com a vulnerabilidade “CVE-2002-1383”, e a “CVE-2002-1383” for explorada em conjunto com a “CVE-2002-1367”, então a vulnerabilidade “CVE-2002-1384” também é explorada. Assim, com a consulta a seguir, é possível correlacionar todas essas vulnerabilidades.

[7] `http://distro.conectiva.com/atualizacoes/index.php?id=a&anuncio=000702`. Acesso em 26/07/2006.

```
Questão de Competência 3: Quais vulnerabilidades são exploradas em conjunto com a vulnerabilidade
CVE-2002-1384?

Consulta 3:
Prefix ab: <http://a.com/SecurityIncidentOntology#>
Select Distinct ?description ?cvecode
Where {?x ab:exploredwithVulnerability ?a;
ab:description ?description;
ab:has_cvecode ?cvecode.
?a ab:has_cvecode "CVE-2002-1384"} Order by ?x
```

6.3 Gerenciamento com o uso da OntoSec

A OntoSec tem por finalidade apoiar os administradores de segurança em sua tarefa de gerenciar os incidentes que podem ocorrer diariamente em um sistema computacional. Esse apoio se dá pela possibilidade de armazenar em um formato único as informações de incidentes de segurança geradas pelas diferentes ferramentas de gerenciamento que podem ser utilizadas em uma organização, como mostra a Figura 6.4. Com as informações de incidentes armazenadas em uma única "ferramenta", os administradores podem tomar decisões com mais eficiência e eficácia. Além disso, à medida que os administradores conhecem melhor seus cenários de incidentes, eles conseguem prevenir futuros ataques, protegendo de maneira mais efetiva seus sistemas.

A Figura 6.4 mostra os níveis de uma organização e onde a OntoSec está inserida[8]. No nível operacional estão as diversas ferramentas de gerenciamento que geram diferentes formatos de alertas de segurança. No nível de conhecimento, a OntoSec define uma estrutura padronizada para armazenar os diferentes formatos desses alertas, gerando uma base de conhecimento de incidentes de segurança. Essa base alimenta um sistema de gerenciamento de incidentes no nível gerencial.

No nível gerencial, os administradores podem questionar a ontologia para saber, por exemplo, a taxa de incidentes relacionados a uma determinada vulnerabilidade, a taxa de incidentes e tipos de incidentes que ocorrem em determinadas épocas do ano, procurando correlacionar tipos de códigos maliciosos com os incidentes que acontecem nessas épocas, e a freqüência com que os serviços críticos (e-mails, arquivos, Web) são atingidos por incidentes de segurança.

Diferentemente dos trabalhos apresentados no Capítulo 3, Seção 3.4, a OntoSec trata as informações sobre incidentes de segurança em um nível gerencial. A partir dessas informações e do conhecimento adquirido pelos administradores, os diretores de uma organização podem criar novas regras nas políticas de segurança.

[8]Laudon e Laudon (2002) apresentam detalhes sobre os níveis de uma organização.

Figura 6.4: OntoSec como apoio às decisões gerenciais de segurança.

6.4 Considerações Finais

A modelagem do sistema representa um importante passo no gerenciamento de incidentes, pois o sistema possibilitará a interoperabilidade entre diferentes ferramentas, armazenando em um formato único as informações sobre os incidentes. Além disso, o sistema mostra o potencial da ontologia em representar informações sobre incidentes, gerando conhecimento para que os administradores de segurança possam tomar decisões de maneira mais eficiente e eficaz. Além disso, o sistema mostra o potencial da ontologia em representar e correlacionar informações sobre incidentes de segurança.

A principal dificuldade na implementação do sistema proposto está, principalmente, na necessidade de se entender os diferentes padrões utilizados pelas ferramentas de segurança para que o mapeamento seja realizado. Assim, à medida que novas ferramentas são instaladas nos ambientes de rede, novos mapeamentos são necessários. Portanto, é importante desenvolver o processo de mapeamento utilizando módulos, como já foi feito no processo de validação da OntoSec.

Capítulo

7

Conclusão e Trabalhos Futuros

A quantidade informações de segurança está crescendo exponencialmente, tornando as tarefas de manipular e gerenciar essas informações bastante custosas. Diversas ferramentas de gerenciamento de segurança podem ser utilizadas para auxiliar os administradores nessas tarefas. Essas ferramentas podem monitorar tudo que entra e saí de uma intranet, podem monitorar o tráfego interno da rede para saber o que está acontecendo e detectar possíveis ataques, podem varrer arquivos em busca de códigos maliciosos, podem criar filtros de *emails* evitando *spams*, ou podem varrer uma rede em busca de vulnerabilidades nos sistemas. Essas ferramentas coletam e armazenam informações de segurança em formatos próprios e diferentes.

Essa falta de um formato único, faz com que o trabalho dos administradores de segurança fique ainda mais difícil, pois ele deve ser capaz de entender todos esses formatos para identificar, por exemplo, quando há um ataque ou uma invasão em andamento. Além disso, o administrador deve se utilizar de seu conhecimento tácito para identificar que um determinado incidente de segurança está relacionado com um incidente prévio ou com uma vulnerabilidade, por exemplo.

Algumas iniciativas apresentadas ao longo deste trabalho realizam esforços no sentido de estabelecer um formato para armazenar informações de segurança. No entanto, essas iniciativas são empregadas em um nível operacional, focalizando ferramentas de segurança específicas, como IDSs, ou informações específicas, como vulnerabilidades. Em um nível gerencial, no qual os administradores têm que lidar com informações provenientes de diver-

sas fontes, faz-se necessário disponibilizar uma "ferramenta" que possa unificar todas essas informações em um formato único, facilitando a tomada de decisão dos administradores.

Nesse contexto, ontologias podem ser utilizadas para definir um formato único para representar e armazenar informações a respeito de um domínio de conhecimento. Assim, este trabalho desenvolveu a Ontologia de Incidentes de Segurança, a OntoSec, para representar e estruturar informações sobre incidentes de segurança.

7.1 Principais Contribuições

Os administradores lidam constantemente com problemas de segurança e tomam decisões a todo o momento. Diante da grande quantidade de informações de segurança geradas por diferentes ferramentas, a tomada de decisões deve ser rápida, pois um incidente de segurança pode atingir grandes proporções em pouco tempo. Para que a tomada de decisão seja mais eficiente, é necessário que essas informações estejam unificadas em uma única "ferramenta", facilitando o trabalho dos administradores. Assim, a ontologia desenvolvida é uma "ferramenta" que pode contribuir para melhorar a eficiência na tomada de decisões, minimizando o impacto que problemas de segurança podem causar em uma organização.

Este trabalho também possui as seguintes contribuições:

1. Reúso de ontologias. A OntoSec integrou parte da ontologia desenvolvida por Brandão (2004) para representar informações sobre vulnerabilidades, mostrando sua capacidade em reutilizar conhecimento de outras ontologias do domínio de segurança. Além disso, conceitos representados pela OntoSec podem ser reutilizados por uma ontologia de mais alto nível (uma *upper-ontology*), baseada em políticas de segurança de uma organização. A OntoSec pode ser integrada, por exemplo, à *Core Ontology* desenvolvida por Schumacher (2003). Essa ontologia define conceitos como *asset*, *attack*, *attacker* e *vulnerability* que estão representados, e estendidos, na OntoSec.

2. Mapeamento automático dos alertas de segurança gerados por uma ferramenta de detecção de intrusão, o **Snort**, para a OntoSec. Esse mapeamento foi um passo importante na tarefa de facilitar a interoperabilidade entre diferentes ferramentas de segurança, permitindo unificar as informações geradas por essas ferramentas em um único formato definido pela ontologia.

3. Avaliação da OntoSec, mostrando o seu potencial em representar informações sobre incidentes de segurança. O processo de avaliação consistiu em:

   - Validar a ontologia com um conjunto de alertas de segurança gerados por uma ferramenta de detecção de intrusão, o **Snort**, mostrando que é possível realizar uma correlação entre incidentes de segurança e gerar conhecimento que pode ser compartilhado e reutilizado por administradores de segurança.
   - Avaliar a ontologia de acordo com critérios como consistência, expressividade, capacidade de expansão, utilidade, entre outros, utilizando para alguns desses critérios a máquina de inferência Pellet.

4. Modelagem de um sistema de gerenciamento de incidentes de segurança baseado na OntoSec. Esse sistema tem como principais objetivos: (i) apoiar os administradores nas tomadas de decisão sobre problemas de segurança em um nível gerencial; (ii) facilitar a correlação entre os incidentes; e (iii) possibilitar a interoperabilidade entre diferentes ferramentas de segurança.

Essas contribuições resultaram em quatro trabalhos publicados em eventos internacionais e nacionais, uma palestra, e um artigo submetido a uma revista internacional na área de segurança.

- ***The Evaluation Process of a Security Incident Ontology***. Esse artigo descreve as atividades realizadas durante o processo de avaliação da OntoSec (Martimiano e Moreira, 2006a). Aceito para publicação no *2nd Workshop on Ontologies and their Applications (WONTO2006)* - outubro de 2006.

- ***Using ontologies to assist security management***. Esse resumo descreve as principais características da OntoSec e como ela pode ser utilizada para auxiliar no gerenciamento de segurança (Martimiano e Moreira, 2006b). Aceito como resumo estendido no *VI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg2006)*, agosto de 2006.

- ***An OWL-based Security Incident Ontology***. Esse pôster descreve o uso da OWL no desenvolvimento da OntoSec (Martimiano e Moreira, 2005). Publicado nos anais da *8th International Protégé Conference* em julho de 2005[1].

[1]No evento de 2006, o *9th International Protégé Conference*, o pôster entitulado *The Value of Semantics on the Management of Security Incidents* foi aceito, mas não foi publicado, pois nenhum dos autores pôde ir apresentar o trabalho.

- ***Towards a security network incident ontology to ease the security knowledge management***. Esse artigo descreve a idéia e o desenvolvimento iniciais da ontologia de incidentes de segurança (Martimiano et al., 2004). Publicado nos anais do *3rd International Information and Telecommunication Techniques Symposium (I2TS2004)* em dezembro de 2004.

- ***Gerenciando Conhecimento em Sistemas de Segurança Computacional: O Uso de Ontologia em Alertas de Vulnerabilidades***. Palestra proferida no *Workshop de Tratamento de Incidentes de Segurança (WTIS2004), VI Simpósio de Segurança em Informática (SSI2004)* em novembro de 2004.

- ***Ontologies and Information Security Governance***. Esse artigo descreve os esforços do grupo no uso de ontologias para representar informações de segurança e para facilitar a governança da segurança da informação em uma organização. Submetido para a revista *Computers & Security* da Elsevier em outubro de 2005.

As potenciais publicações deste trabalho são: (i) artigo descrevendo a proposta do sistema de gerenciamento de incidentes de segurança baseado na OntoSec; (ii) artigo descrevendo as dificuldades enfrentadas na fase de Conceituação de uma ontologia, principalmente nas decisões de como os conceitos são representados, como classes, atributos ou instâncias; (iii) artigo descrevendo a importância da interoperabilidade entre ferramentas de segurança no nível gerencial de uma organização.

7.2 Propostas de Trabalhos Futuros

As contribuições deste trabalho de doutorado representam um ponto importante na discussão do uso de ontologias para representar e correlacionar informações sobre incidentes de segurança, e para facilitar a interoperabilidade entre diferentes ferramentas de gerenciamento de segurança. Além disso, o processo de validação e avaliação da ontologia também tem uma grande contribuição na área de pesquisa sobre avaliação de ontologias.

A partir dessas contribuições, as principais propostas para trabalhos futuros são:

1. Estender a OntoSec para representar outros conceitos relacionados a incidentes de segurança. Por exemplo, incluir quais medidas são necessárias para se recuperar de um incidente, criando uma classe para representar o conceito *Counter Measure*. Também seria interessante estabelecer responsabilidades sobre a implantação e execução dessas medidas, criando uma classe *Security Administrator*.

2. Desenvolver novas ontologias a partir dos conceitos representados na OntoSec, tais como: uma ontologia de vírus, uma ontologia de *worms*, uma ontologia de sistemas de detecção de intrusão ou uma ontologia de *firewalls*. O desenvolvimento dessas ontologias contribuirá para que o domínio de Segurança possa ser, aos poucos, representado por meio de ontologias.

3. Mapear incidentes de segurança reais para a OntoSec. Esse mapeamento validará uma quantidade maior de conceitos e propriedades que estão representados na OntoSec. Os incidentes registrados pelo CSIRT USP já estão disponíveis para a realização dessa tarefa.

4. Desenvolver os módulos de inferência, de visualização e de relatórios do sistema de gerenciamento de incidentes de segurança, tornando-o operacional em um ambiente de redes de uma organização.

5. Incorporar alertas de segurança gerados por outras ferramentas, tais como *firewalls* e programas antivírus, aprimorando o Módulo de Mapeamento e a interoperabilidade entre as ferramentas.

6. Aprimorar os módulos de consultas e inserção do sistema de gerenciamento, melhorando o processo de validação da OntoSec.

7. Aprimorar o processo de avaliação, utilizando outras máquinas de inferências para, por exemplo, realizar uma análise comparativa entre os resultados obtidos pela Pellet com os resultados das outras máquinas, como Racer e FaCT++, melhorando o processo de correlação entre incidentes de segurança.

Referências Bibliográficas

ABNT Tecnologia da Informação - código de prática para a gestão da segurança da informação. Associação Brasileira de Normas Técnicas - ABNT, 2001.

Allen, J.; Christie, A.; Fithen, W.; McHugh, J.; Pickel, J.; Stoner, E. *State of the practice of intrusion detection technologies.* Relatório Técnico, Software Engineering Institute, Carnegie Mellon University, TR CMU/SEI-99-TR-028, ESC-99-028, 2000.

Ambrósio, D. R. *Métodos alternativos de reconhecimento de padrões para sistemas de detecção de intrusão.* Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação - ICMC, Universidade de São Paulo - USP, São Carlos - São Paulo, 2002.

Balasubramaniyan, J. S.; Fernandez, J. O. G.; Isacoff, D.; Spafford, E.; Zamboni, D. *An architecture for intrusion detection using autonomous agents.* Relatório Técnico, Department of Computer Sciences, Purdue University, COAST Laboratory TR 98/05, 1998.

Bechhofer, S.; Harmelen, F. v.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; Stein, L. A. OWL web ontology language reference. [*On-line*], http://www.w3.org/TR/owl-ref/.

Beckett, D. RDF/XML syntax specification (revised). [*On-line*], http://www.w3.org/TR/rdf-syntax-grammar/.

Bernardes, M. C. *Modelagem de sistemas de segurança computacional como sistemas de informação.* Qualificação de Doutorado, Instituto de Ciências Matemáticas e de Computação - ICMC, Universidade de São Paulo - USP, São Carlos - São Paulo, 2002.

Berners-Lee, T.; Fielding, R.; Irvine, U. C.; Masinter, L. Uniform Resource Identifiers (uri): Generic syntax - rfc2396. [*On-line*], http://rfc.net/rfc2396.html.

Berners-Lee, T.; Hendler, J.; Lassila, O. The Semantic Web. *Scientific American*, v. 284, n. 5, p. 34–43, 2001.

Brandão, A. J. S. *Uso de ontologia para classificação de vulnerabilidades em sistemas computacionais*. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação - ICMC, Universidade de São Paulo - USP, São Carlos - São Paulo, 2004.

Brandão, A. J. S.; Martimiano, L. A. F.; Moreira, E. S. O uso de ontologia em alertas de vulnerabilidades. In: *Anais do Quarto Workshop de Segurança em Sistemas Computacional - Vigésimo Segundo Simpósio Brasileiro de Redes de Computadores*, 2004.

Brank, J.; Grobelnik, M.; Mladeni, D. A survey of ontology evaluation techniques. In: *Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD05)*, 2005.

Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; Yergeau, F. EXtensible Markup Language (XML) 1.0 (3rd edition). [*On-line*], http://www.w3.org/TR/REC-xml/.

Brewster, C.; Alani, H.; Dasmahapatra, S.; Wilks, Y. Data driven ontology evaluation. In: *Proceedings of the International Conference on Language Resources and Evaluation*, 2004.

Brickley, D.; Guha, R. V. RDF vocabulary description language 1.0: RDFSchema. [*On-line*], http://www.w3.org/TR/rdf-schema/.

Bézivin, J. Who's afraid of ontologies? [*On-line*], http://www.metamodel.com/oopsla98-cdif-workshop/bezivin1/.

CC09 Common criteria for information technology security evaluation. Preliminary DRAFT Version 0.9, 1994.

Chandrasekaran, B.; Josephson, J. R.; Benjamins, V. R. What are ontologies, and why do we need them? *IEEE Intelligent Systems - Special Issue on Ontologies*, v. 14, n. 1, p. 20–26, 1999.

Chapman, D. B.; Zwick, E. D. *Building internet firewalls*. O'Reilly & Associates, 517 p., 1995.

Chen, C. S.; Tseng, S. S.; Liu, C. L.; Ou, C. H. Building a DNS ontology using Methontology and Protégé-2000. In: *Proceedings of 2002 International Computer Symposium Workshop on Artificial Intelligence (ICS02)*, 2002, p. 1853–1860.

Chintan, P.; Kaustubh, S.; Yugyung, L.; Park, E. K. Ontokhoj: A semantic web portal for ontology searching, ranking and classification. In: *Proceedings of the Fifth ACM International Workshop on Web Information and Data Management*, 2004, p. 58–61.

Corcho, O.; Gómez-Pérez, A.; González-Cabero, R.; Suárez-Figueroa, M. C. Odeval: A tool for evaluating RDF(S), DAML+OIL and OWL concept taxonomies. In: *Proceedings of the First IFIP Conference on Artificial Intelligence Applications and Innovations (AIAI04)*, 2004, p. 369–382.

Curry, D.; Debar, H.; Feinstein, B. Intrusion detection message exchange format. [*On-line*], internet draft. http://www3.ietf.org/proceedings/05mar/IDs/draft-ietf-idwg-idmef-xml-14.txt.

Das, A.; Wu, W.; McGuinness, D. L. *Industrial strength ontology management*. Relatório Técnico, Stanford Knowledge Systems Laboratory, TR KSL-01-09, 2001.

Deepsia Dynamic on-line internet purchasing system based on intelligent agent. [*On-line*], http://www.deepsia.com/br.

Dekker, M. Security of the internet. In: *The Froehlich/Kent Encyclopedia of Telecommunications*, 1997, p. 231–255.

DoD Department of defense trusted computer system evaluation criteria. DOD 5200.28-STD, 1985.

Donner, M. Toward a security ontology. *IEEE Security & Privacy Magazine*, v. 1, n. 3, p. 6–7, 2003.

Elmasri, R.; Navathe, S. B. *Fundamentals of database systems*. 4th ed. Addison-Wesley, 1030 p., 2003.

Eulália, L. A. S.; Moreira, E. S.; H., R.; Carvalho, A. Using ontologies for intelligent information retrieval in an e-commerce application case study. In: *Proceedings of the Fourth European Conference on Product and Process Modeling in the Building and Related Industries*, 2002, p. 277–284.

Everett, J. O.; Borrow, D. G.; Stolle, R.; Crouch, R.; Paiva, V.; Condoravdi, C.; Van Den Berg, M.; Polanyi, L. Making ontologies work for resolving redundancies across documents. *Communications of the ACM*, v. 45, n. 2, p. 55–60, 2002.

Falbo, R. A. *Integração de conhecimento em um ambiente de desenvolvimento de software.* Tese de Doutoramento, Coppe - Universidade Federal do Rio de Janeiro, Rio de Janeiro - Rio de Janeiro, 1998.

Fallside, D. C.; Walmsley, P. XML schema part 0: Primer 2nd edition. [*On-line*], http://www.w3.org/TR/xmlschema-0/.

FC1 Federal criteria for information technology security: Vol. i, protection profile development; vol. ii, registry of protection profiles, version 1.0. National Institute of Standards and Technology (NIST) and National Security Agency (NSA), 1992.

Fensel, D.; Horrocks, I.; van Harmelen, F.; Decker, S.; Erdmann, M.; Klein, M. C. A. Oil in a nutshell. In: *Proceedings of the Twelveth European Workshop on Knowledge Acquisition, Modeling and Management - EKAW00*, Lecture Notes In Computer Science, 2000, p. 1–16.

Fernández, M. A.; Gómez-Pérez, A.; Juristo, N. Methontology: From ontological art towards ontological engineering. In: *Proceedings of the AAAI Spring Symposium Series*, 1997, p. 33–40.

Filman, R.; Linden, T. Safebots: A paradigm for software security controls. In: *Proceedings of the ACM New Security Paradigms Workshop*, 1996, p. 45–51.

Fonseca, F.; Egenhofer, M.; Borges, K. Ontologias e interoperabilidade semântica entre sigs. In: *Anais do Segundo Workshop Brasileiro em Geoinformática - GeoInfo2000*, 2000, p. 45–52.

Garfinkel, S.; Spafford, G. *Practical Unix and internet security.* 2nd ed. O'Reilly & Associates, 971 p., 1996.

Gennari, J.; Musen, M. A.; Fergerson, R. W.; Grosso, W. E.; Crubezy, M.; Eriksson, H.; Noy, N. F.; Tu, S. W. The evolution of protégé: An environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, v. 58, n. 1, p. 89–123, 2003.

GÓMEZ-PÉREZ, A. Evaluation of taxonomic knowledge in ontologies and knowledge bases. In: *Banff Knowledge Acquisition for Knowledge-Based Systems (KAW99)*, 1999, p. 6.1.1–6.1.18.

GÓMEZ-PÉREZ, A.; CORCHO, O. Ontology languages for the Semantic Web. *IEEE Intelligent Systems*, v. 17, n. 1, p. 54–60, 2002.

GÓMEZ-PÉREZ, A.; FERNÁNDEZ, M.; VICENTE, A. J. Towards a method to conceptualize domain ontologies. In: *Proceedings of the Workshop on Ontological Engineering (ECAI96)*, 1996, p. 41–52.

GÓMEZ-PÉREZ, A.; JURISTO, N.; MONTES, C.; PAZOS, J. Ingeniería del conocimiento: Diseño y construcción de sistemas expertos. Ceura, Madri, Spain, 1997.

GOLLMANN, D. *Computer security.* John Wiley & Sons, 320 p., 1999.

GRUBER, T. R. Towards principles for the design of ontologies used for knowledge sharing. In: *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers, 1993.

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, v. 43, n. 5/6, p. 907–928, 1995.

GRUNINGER, M.; FOX, M. S. Methodology for design and evaluation of ontologies. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI98) - Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

GRUNINGER, M.; LEE, J. Ontology: applications and design. *Communications of the ACM*, v. 45, n. 2, p. 39–41, 2002.

HARMELEN, F. V.; PATEL-SCHNEIDER, P. F.; HORROCKS, I. Reference description of the DAML+OIL ontology markup language. [*On-line*], http://www.daml.org/2001/03/reference.

HEIDARI, M. Malicious codes in depth. [*On-line*], http://www.securitydocs.com/library/2742.

HERRERA, J. A. F.; MARTINS JÚNIOR, J.; MOREIRA, E. S. A model for data manipulation and ontology navegation in deepsia project. In: *Proceedings of the First Seminar on Advanced Research in Electronic Business (EBR02)*, 2002.

Holsapple, C. W.; Joshi, K. D. A collaborative approach to ontology design. *Communications of the ACM*, v. 45, n. 2, p. 42–47, 2002.

Horridge, M.; Knublauch, H.; Rector, A.; Stevens, R.; Wroe, C. *A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools*. Relatório Técnico, University Of Manchester, 2004.

Horrocks, I.; Fensel, D.; Broekstra, J.; Decker, S.; Erdmann, M.; Goble, C.; Klein, M.; Staab, S.; Studer, R.; Motta, E. The ontology inference layer oil. [*On-line*], http://wwwis.win.tue.nl/j̃broekst/papers/oil.pdf.

Houaiss, A. *Houaiss - dicionário da língua portuguesa*. Editora Objetiva, 2003.

Howard, J. D. *An analysis of security incidents on the internet: 1989-1995*. Tese de Doutoramento, Carnegie Mellon University - Institute of Technology, 1997.

Howard, J. D.; Longstaff, T. A. *A common language for computer security incidents*. Relatório Técnico, Sandia National Laboratories, 1998.

IEEE Ieee standard for developing software life cycle processes. IEEE Computing Society, 1996.

ISO Information processing systems - open systems interconnection - basic reference model - part 2 - security architecture. ISO International Eletrotechnical Committee - International Standard 7498-2, 1989.

J., P. *Conceptualización*. Dissertação de Mestrado, Facultad de Informática de Madrid - Universidad Politécnica de Madrid, Madrid - Spain, 1995.

Karp, R.; Chaudhri, V.; Thomere, J. XOL: An XML-based ontology exchange language(version 0.4. [*On-line*], http://www.ai.sri.com/p̃karp/xol.

Kent, R. E. Conceptual knowledge markup language: The central core. In: *In the Electronic Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management (KAW99)*, http://sern.ucalgary.ca/ksi/kaw/kaw99/papers/Kent1/CKML.pdf, 1999.

Klyn, G.; Carroll, J. J. Resource Description Framework (RDF): concepts and abstract syntax. [*On-line*], http://www.w3.org/TR/rdf-concepts/.

Kumar, S. *Classification and detection of computer intrusions*. Tese de Doutoramento, Computer Sciences Departament - Purdue University, 1995.

LARMAN, G. *Utilizando UML e padrões.* Bookman, 607 p., Tradução da segunda edição, 2004.

LAUDON, K. C.; LAUDON, J. P. *Management Information Systems - managing the digital firm.* 7th ed. Prentice Hall, 547 p., 2002.

LIMA, C.; FIÉS, B.; ZARLI, A.; BOURDEAU, M.; WETHERILL, M.; REZGUI, Y. Towards an ifc-enable ontology for building and construction industry; the e-cognos approach. In: *Proceedings of the CISEMIC*, 2002.

LOZANO-TELLO, A.; GÓMEZ-PÉREZ, A. Ontometric: A method to choose the appropriate ontology. *Journal of Database Management*, v. 15, n. 2, p. 1–18, 2004.

LÓPEZ, M. F. Chemicals: Ontologia de elementos químicos. Faculdad de Informática de la Universidad Politécnica de Madrid. Final-Year Project, 1996.

LÓPEZ, M. F.; GÓMEZ-PÉREZ, A.; JURISTO, N.; J., P.-S. Building a Chemical Ontology using Methontology and the Ontology Design Environment. *IEEE Intelligent Systems*, v. 14, n. 1, p. 37–46, 1999.

LUKE, S.; HEFLIN, J. Proposed specification, SHOE project, shoe 1.01. [*On-line*], http://www.cs.umd.edu/projects/plus/SHOE/spec101.htm.

MAEDCHE, A.; STAAB, S. Measuring similarity between ontologies. In: *Proceeding of the Thirteenth European Conference on Knowledge Acquisition and Management (EKAW02)*, 2002, p. 251–263.

MANN, D. E.; CHRISTEY, S. M. Towards a common enumeration of vulnerabilities. [*On-line*], http://cve.mitre.org.

MARTIMIANO, L. A. F.; BRANDÃO, A. J. S.; MOREIRA, E. S. Towards a security network incident ontology to ease the security knowledge management. In: *Proceedings of the Third International Information and Telecommunication Techniques Symposium (I2TS2004)*, 2004, p. 88–95.

MARTIMIANO, L. A. F.; MOREIRA, E. S. An owl-based security incident ontology. In: *Proceedings of the Eighth International Protégé Conference*, Poster, 2005, p. 43–44.

MARTIMIANO, L. A. F.; MOREIRA, E. S. The evaluation process of a computer security incident ontology. In: *Proceedings of the Second Workshop on Ontologies and their Applications (WONTO06)*, 2006a.

MARTIMIANO, L. A. F.; MOREIRA, E. S. Using ontologies to assist security management. In: *Proceedings of the Sixth Brazilian Symposium on Information and Computer System Security (SBSeg06)*, Short Paper, 2006b.

MARTIN, R. A. Managing vulnerabilities in network systems. *IEEE Computer*, v. 34, n. 11, p. 32–38, 2001.

MARTIN, R. A.; CHRISTEY, S.; BAKER, D. *A progress report on the cve initiative.* Relatório Técnico, The MITRE Corporation, http://www.cve.mitre.org/docs/docs2002/prog-rpt_06-02/index.html, 2003.

MCGUINNESS, D. L.; HARMELEN, F. V. OWL web ontology language - overview. [*On-line*], http://www.w3.org/TR/owl-features/.

MÓDULO Nona pesquisa nacional de segurança da informação. Módulo Security Solution, 2003.

MENZIES, T. Cost benefits of ontologies. *Intelligence*, v. 10, n. 3, p. 26–32, 1999.

MOURA, A. M. C. A web semântica: Fundamentos e tecnologias. In: *Anais do VI Congresso Internacional de Ciencias de la Computación*, 2001, p. 46–82.

NCSC Glossary of computer security terms. National Computer Security Center - Trusted Network (NCSC-TG-004), 1988.

NOY, N. F.; MCGUINNESS, D. L. *Ontology development 101: A guide to create your first ontology.* Relatório Técnico, Knowledge Systems Laboratory - Stanford University, TR KSL-01-05, 2001.

OPPLIGER, R. Internet security: firewalls and bey. *Communication of the ACM*, v. 40, n. 5, p. 92–102, 1996.

PACHECO, R. C. S.; KERN, V. M. Uma ontologia comum para a integração de bases de informação e conhecimento sobre ciência e tecnologia. *Revista Ciência da Informação*, v. 30, n. 3, p. 56–63, 2001.

PORZEL, R.; MALAKA, R. A task-based approach for ontology evaluation. In: *Proceedings of the Workshop on Ontology Learning and Population (ECAI2004) - Sixteenth European Conference on Artificial Intelligence*, 2004, p. 9–16.

PRUD'HOMMEAUX, E.; SEABORNE, A. SPARQL - query language for RDF. [*On-line*], http://www.w3.org/TR/rdf-sparql-query/.

RASKIN, V.; HEMPELMANN, C. F.; TRIEZENBERG, K. E.; NIRENBURG, S. Ontology in information security: A useful theoretical foundation and methodology tool. In: *Proceedings of the Workshop on New Security Paradigms*, 2001.

RAY, J.; ANONYMOUS *Maximum linux security.* 2nd ed. Sams Publishing, 896 p., 2001.

ROSE, M. RFC 1225: Post office protocol. Network Working Group, 1991.

SANTOS, M. L. *Mapeamento dos dados de uma ferramenta de segurança em uma ontologia de incidentes de segurança.* Relatório Técnico, Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo, bolsa PIBIC, 2006.

SCHNEIER, B. *Secrets & lies - digital security in a networked world.* John Wiley & Sons, 412 p., 2000.

SCHUMACHER, M. Toward a security core ontology. In: *Security Engineering with Patterns - Origins, Theoretical Model and New Applications*, Springer Verlag, p. 87–96, lectures Notes in Computer Science (LNCS 2754), 2003.

SEABORNE, A. RDQL - a query language for RDF. [*On-line*], http://www.w3.org/Submission/RDQL/.

SHIREY, R. RFC 2828: Internet security glossary. GTE-BBN Technologies, 2000.

SIRIN, E.; PARSIA, B.; CUENCA, B. G.; KALYANPUR, A.; KATZ, Y. Pellet: A practical OWL-DL reasoner. Submitted for publication to Journal of Web Semantics, 2006.

SMITH, M. K.; WELTY, C. MCGUINNESS, D. L. OWL web ontology language - guide. [*On-line*], http://www.w3.org/TR/owl-guide/.

SOARES, L. F. G.; LEMOS, G.; COLCHER, S. *Redes de computadores: das LANs, MANs e WANs às redes ATM.* Segunda ed. Editora Campus - Rio de Janeiro, 704 p., 1995.

TANENBAUM, A. S. *Sistemas operacionais.* Prentice Hall, 695 p., Tradução da segunda edição, 2003.

TEMPICH, C.; VOLZ, R. Towards a benchmark for semantic web reasoners - an analysis of the daml ontology library. In: *Proceedings of Evaluation of Ontology-based Tools (EON03) at Second International Semantic Web Conference (ISWC03)*, 2003, p. 4–15.

TrendMicro Glossary of virus terms. [*On-line*], http://www.trendmicro.com/en/security/general/glossary/overview.htm.

Undercoffer, J.; Joshi, A.; Pinkston, J. Modeling computer attacks: An ontology for intrusion detection. In: *Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection (RAID03)*, 2003, p. 113–135.

Undercoffer, J.; Pinkston, J. Modeling computer attacks: a target-centric ontology for intrusion detection. In: *Proceedings of the Center for Architectures for Data-driven Information Processing Research Symposium (CADIP02)*, 2002.

Undercoffer, J.; Pinkston, J.; Joshi, A.; Finin, T. A target-centric ontology for intrusion detection. In: *Proceedings of the Workshop on Ontologies and Distributed Systems. Eighteenth International Joint Conference an Artificial Intelligence*, 2004.

Uschold, M. Building ontologies: Towards a unified methodology. In: *Proceeding of the Sixteenth Annual Conference of the British Computer Society Specialist Group on Expert Systems*, 1996.

Uschold, M.; Gruninger, M. Ontologies: principles, methods and applications. *Knowledge Engineering Review*, v. 11, n. 2, p. 93–155, 1996.

Uschold, M.; King, M. Towards a methodology for building ontologies. In: *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing - IJCAI95*, 1995.

Vieira, R.; Santos, D. A.; Silva, D. M.; Santana, M. R. Web semântica: ontologias, lógica de descrição e inferência. Tutorial do XI Simpósio Brasileiro de Sistemas Multimídia e Web (WEBMEDIA05), 2005.

Welty, C. A.; ; Guarino, N. Supporting ontological analysis of taxonomic relationships. *Data Knowledge Engineering*, v. 39, n. 1, p. 51–74, 2001.

West-Brown, M. J.; Stikvoort, D.; Kossakowski, K. *Handbook for computer security incident response teams.* Carnegie Mellon - Software Engineering Institute, 496 p., CMU/SEI-98-HB-001, 1998.

Zuñiga, G. L. Ontology: Its transformation from philosophy to information systems. In: *Proceedings of the Second International Conference on Formal Ontology in Information Systems - FOIS01*, 2001, p. 187–197.

Apêndice

A

Vocabulário de Conceitos e Relacionamentos da OntoSec

A.1 Vocabulário de Conceitos

A

Access. *The ability and means to communicate with or otherwise interact with a system in order to use system resources to either handle information or gain knowledge of the information the system contains* (Shirey, 2000).

Access Validation Error. *Vulnerability in a system's access validation.*

Account Compromise. *Is the unauthorized use of a computer account by someone other than the account owner, without involving system-level or root-level privileges (privileges a system administrator or network manager has). An account compromise might expose the victim to serious data loss, data theft, or theft of services. The lack of root-level access means that the damage can usually be contained, but a user-level account is often an entry point for greater access to the system* (Dekker, 1997).

Active Attack. *Attack that attempts to alter system resources or to affect their operation* (Shirey, 2000).

Adware. *Software that displays advertising banners on Web browsers. Often create unwanted effects on a system, such as annoying popup ads and, in some instances, the degradation in either network connection or system performance* (TrendMicro, 2006).

Agent. *Entity that perfoms attacks in order to cause security incidents.*

Address Resolution Protocol (ARP). *Method for finding a host's MAC address when only its IP address is known. The sender broadcasts an ARP packet containing the Internet address of another host and waits for it to respond with its MAC (Medium Access Control) address* (Tanenbaum, 2003).

Antivirus. *Program that searches for malicious codes in files and removes them* (`http://www.webopedia.com/TERM/A/antivirus_program.html`).

Asset. *Anything that has a value in a computational system in the organization and can be attacked during a security incident* (`http://www.modulo.com.br`).

Attack. *Assault on system security that derives from an intelligent threat, i.e., an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system* (Shirey, 2000).

Authorized User. *User who has a specific right or permission to do something in a system* (CC09, 1994).

B

Backdoor. *1. Tool that (a) provides access to a system and its resources by other than the usual procedure, (b) was deliberately left in place by the system's designers or maintainers, and (c) usually is not publicly known.* ***Synonymous***: *Trap Door.* (Shirey, 2000). *2. Program that opens computers for access by remote systems* (TrendMicro, 2006).

Bandwidth. *The amount of information that can be passed through it in a given amount of time, usually expressed in bits per second* (DoD, 1985).

Bridge. *Computer that is a gateway between two networks, usually two LANs, at OSI layer 2 that use the same protocol* (Shirey, 2000).

Buffer Overflow. *Attack that attempts to cause a failure in a computer system or other data processing entity by providing more input than the entity can process properly.* ***Synonymous***: *Flooding; Overloading.* (Shirey, 2000)

C

Client Side. *Vulnerability that is explored in the client side.*

Command. *Collection of primitives used to attack a system.*

Communication Channel. *Means used to pass data or information in the network.*

Configuration Change. *Unauthorized configuration changes in a system.*

Configuration Error. *Vulnerability in a system's configuration.*

Consequence. *Damages caused to a system when security incidents had happened.*

Context Error. *Vulnerability in a system's context.*

Cookie Poisoning. *Security incident that enables cookies manipulation in order to cause damage.*

Correction. *Patches used to fix a vulnerability.*

Cross Site Scripting (XSS). *Malicious HTML embedded in web client requests* (Shirey, 2000).

Cryptography Module. *Principles, means and methods for rendering information unintelligible, and for restoring encrypted information to intelligible form* (`http://secinf.net/policy_and_standards/US_Department_Of_Defense_Glossary_of_Computer_Security_Terms.html`).

CVE (Common Vulnerabilities and Exposures) Code. *Standardize code for a publicly known vulnerabilities and security exposures* (Mann e Christey, 1999).

D

Data. *File or process that can be processed by or stored in a computer system, and can be compromised by a security incident.*

Database Manager. *Collection of programs that enables users to create and maintain a database. General-purpose software system that facilitates the processes of defining, constructing, manipulating and sharing databases among varios users and applications* (Elmasri e Navathe, 2003).

Deface. *Web pages falsification* (Shirey, 2000).

Denial of Service (DoS). *1. The prevention of authorized access to a system resource or the delaying of system operations and functions* (Shirey, 2000). *2. The goal of denial-of-service attacks is not to gain unauthorized access to machines or data, but to prevent legitimate users of a service from using it. A denial-of-service attack can come in many forms. Attackers may "flood" a network with large volumes of data or deliberately consume a scarce or limited resource, such as process control blocks or pending network connections. They may also disrupt physical components of the network or manipulate data in transit, including encrypted data* (Dekker, 1997).

Design Error. *Vulnerability in a system's design.*

Destination Port. *Logical port in the system under attack that was used during a security incident.*

Distributed Denial of Service (DDoS). *The prevention of authorized access to a system resource or the delaying of system operations and functions, using several computers to attack* (Shirey, 2000).

Domain Name Service (DNS) Server . *The main Internet operations database, which is distributed over a collection of servers and used by client software for purposes such as translating a domain name-style host name into an IP address and locating a host that accepts mail for some mailbox address* (Shirey, 2000).

Dynamic Host Configuration Protocol (DHCP). *Client-server networking protocol. A DHCP server provides configuration parameters specific to the DHCP client host requesting, generally, information required by the client host to participate on an IP network. DHCP also provides a mechanism for allocation of IP addresses to client hosts* (`http://en.wikipedia.org/wiki/DHCP`).

E

Exploit. *1. Code used to perform an attack on a computer system, especially one that takes advantage of a particular vulnerability that the system has* (`http://www.whatis.com`). *2. Code that takes advantage of a software vulnerability or security hole. Exploits are often incorporated into malware, which are consequently able to propagate into and run intricate routines on vulnerable computers* (TrendMicro, 2006).

Exception Error. *Vulnerability in a system's exception.*

F

File Server. *Software process that runs on a computer connected to the Internet and serves files to a system users.*

File Transfer Protocol (FTP). *TCP-based, application-layer, Internet Standard protocol for moving data files from one computer to another* (Shirey, 2000).

Firewall. *Internetwork gateway that restricts data communication traffic to and from one of the connected networks (the one said to be "inside" the firewall) and thus protects that network's system resources against threats from the other network (the one that is said to be "outside" the firewall)* (Shirey, 2000).

FTP Server. *Software process that runs on a computer connected to the Internet to respond to FTP requests for moving data files from one computer to another.*

G

Gateway. *Relay mechanism that attaches to two (or more) computer networks that have similar functions but dissimilar implementations and that enables host computers on one network to communicate with hosts on the other; an intermediate system that is the interface between two computer networks* (Shirey, 2000).

H

Hardware. *Any physical component of a computer system* (Shirey, 2000).

Harvesting. *Gathering information through vulnerabilities* ().

High Severity. *Security incident that causes serious problems in a system.*

Hijack. *Security incident that modifies browser behavior, allowing accesses to any pages or sites* (CSIRT USP).

Hijacker. *Malware that modifies browser behaviour, accessing any pages or sites* (CSIRT USP).

Host. *Computer that is attached to a communication network and can use services provided by the network to exchange data with other attached systems.* ***Synonymous***: *Computer, End System.* (Shirey, 2000)

Hub. *Device of convergence where data arrives from one or more directions and is forwarded out in one or more other directions in the network*(http://www.whatis.com).

Human *Agent who is the perpetrator of a computer security incident. Intruders are often referred to as "hacker" or "crackers". While hackers were very technical experts in the early days of computing, this term was later used by the media to refer to people who break into other computer systems. Crackers is based on hackers and the fact that these people crack computer systems and security barriers. Most of the time crackers is used to refer to more notorious intruders and computer criminals* (West-Brown et al., 1998).

Hypertext Transfer Protocol (HTTP). *TCP-based, application-layer, client-server, Internet protocol used to carry data requests and responses in the World Wide Web* (Shirey, 2000).

Hypertext Transfer Protocol Secure (HTTPS). *Using an https: URL indicates that HTTP is to be used, but with additional security measures applied to the transactions* (`http://en.wikipedia.org/wiki/Https`).

I

Input/Output (I/O) Device. *Hardware that provides services to a computational system, such as hard disks and printers.*

Interception. *Vulnerability that is explored by interception.*

Internet Control Message Protocol (ICMP). *Internet standard protocol that is used to report error conditions during IP datagram processing and to exchange other information concerning the state of the IP network* (Shirey, 2000).

Internet Message Access Protocol (IMAP). *Application layer Internet protocol that allows a local client to access email on a remote server* (`http://en.wikipedia.org/wiki/IMAP`).

Internet Protocol (IP). *Method or protocol by which data is sent from one computer to another on the Internet. Each a host on the Internet has at least one IP address that uniquely identifies it from all other computers on the Internet* (Tanenbaum, 2003).

Intrusion Detection System (IDS). *Computational system that inspects all inbound and outbound network activity and identifies suspicious patterns that may indicate a network or system attack from someone attempting to break into or compromise a system* (`http://www.webopedia.com/TERM/i/intrusion_detection_system.html`).

IP Destination. *IP addresses from where a security incident had came.*

IP Source. *IP addresses to where a security incident had happened.*

J

No entry.

K

Keylogger. *Malware that log keyboard activity to gather user information. Keyloggers usually catch and store all keyboard activity - leaving a person or another application to sort through the keystroke logs for valuable information like logon credentials and credit card numbers* (TrendMicro, 2006).

L

Lightweight Directory Access Protocol (LDAP). *Networking protocol for querying and modifying directory services running over TCP/IP* (`http://en.wikipedia.org/wiki/LDAP`).

Local Attack. *Attack that is initiated by an entity inside the system, an entity that is authorized to access system resources but uses them in a way not approved by those who granted the authorization* (Shirey, 2000).

Logic Bomb. *Malicious logic that activates when specified conditions are met. Usually intended to cause denial of service or otherwise damage system resources* (Shirey, 2000).

Loss of Availability. *A system or a system resource is no longer accessible and usable upon demand by an authorized system entity, according to performance specifications for the system. Hindrance of system operation by placing excess burden on the performance capabilities of a system component* (Shirey, 2000).

Loss of Confidentiality. *Desability of a system to perform a required function under stated conditions for a specified period of time only for authorized users* (Shirey, 2000).

Loss of Integrity. *A system can not perform its intended function in an unimpaired manner, and it is not free from deliberate or inadvertent unauthorized manipulation* (NCSC, 1988).

Loss of Privacy. *The right of an entity (normally a person), acting in its own behalf, to determine the degree to which it will interact with its environment, including the degree to which the entity is willing to share information about itself with others, is lost* (Shirey, 2000).

Low Severity. *Security incident that does not cause serious problems in a system.*

M

Mail Server. *Software process that runs on a host computer connected to the Internet for moving electronic mail messages from one.*

Malicious Code. *Security incident caused by a malware.*

Malware. *1. Software that is intentionally included or inserted in a system for a harmful purpose. Programs that exploit or checks for the existence of vulnerabilities or objects of unknown type and purpose found on a compromised host* (Shirey, 2000). *2. Program that performs unexpected or unauthorized, but always malicious, actions.* (TrendMicro, 2006).

Man In the Middle. *Security incident in which the attacker intercepts messages in a public key exchange and then retransmits them, substituting their own public key for the requested one, so that the two original parties still appear to be communicating with each other directly* (`http://searchsecurity.techtarget.com/`).

Medium Severity. *Security incident that causes medium problems in a system.*

Memory. *Electronic holding place for instructions and data that your computer's microprocessor can reach quickly.* ***Synonymous***: *Random Access Memory (RAM).*

Miscellaneous Attack. *Unclassified security incident.*

Misuse. *Security incident that causes a system component to perform a function or a service that is detrimental to system security.*

N

Natural *Agent of the nature that causes a security incident, such as a storm.*

Natural Damage. *Consequence implied when natural problems happen.*

Natural Disaster. *Security incident caused by a natural agent.*

Network Basic Input/Output System (NetBios). *API that allows applications on separate computers to communicate over a local area network* (`http://en.wikipedia.org/wiki/Netbios`).

Network Information Service (NIS) Server. *Client-server directory service protocol for distributing system configuration data such as user and host names between computers on a computer network* (`http://en.wikipedia.org/wiki/Network_Information_Service`).

Non-Self Replicating. *Malware that can not replicate themselves.*

O

Open Proxy. *Internet proxy server which is accessible by unauthorized users, specifically those from elsewhere on the Internet. With an open proxy, however, any user is able to access the machine providing this service, potentially for purposes that violate the law or terms of service* (`http://en.wikipedia.org/wiki/Open_proxy`).

Open Relay. *SMTP email server that allows third-party relay of email messages. By processing mail that is neither for nor from a local user, an open relay makes it possible for an unscrupulous sender to route large volumes of spam.* ***Synonymous***: *Insecure Relay; Third-party Relay.* (`http://www.whatis.com`)

Operating System. *Software responsible for the direct control and management of hardware and basic system operations* (Tanenbaum, 2003).

P

Passive Attack. *Attack that attempts to learn or make use of information from the system but does not affect it* (Shirey, 2000).

Physical *System component that can causes a security incident when fails, as hard disks, routers, switches.*

Physical Access *Vulnerability is explored by physical access in a system.*

Physical Damage. *Consequence implied when a physical component fails.*

Physical Problem. *Security incident caused by a failure of a physical component.*

Pre Condition. *Conditions present in a system that allow a security incident.*

Print Server. *Software process that runs on a computer connected to the Internet and serves printers to a system users*

Processor. *1. Logic circuitry that responds to and processes the basic instructions that drive a computer. The term processor has generally replaced the term CPU (Central Processing Unit)* (`http://searchsmb.techtarget.com/`). *2. Component in a digital computer that interprets instructions and processes data contained in computer programs* (`http://en.wikipedia.org/wiki/Central_processing_unit`).

Pornography. *Security Incident that involves any kind of pornography.*

Port. *1. Logical connection place* (`http://searchnetworking.techtarget.com/`). *2. Endpoint to a logical connection* (`http://www.webopedia.com/`). *3. A connection address specified to allow programs on different computers to communicate* (TrendMicro, 2006).

Post Office Protocol (POP). *Application-layer Internet standard protocol, to retrieve email from a remote server over a TCP/IP connection* (Rose, 1991).

Probe. *Unusual attempts to gain access to a system or to discover information about the system. One example is an attempt to log in to an unused account. Probing is the electronic equivalent of testing doorknobs to find an unlocked door for easy entry. Probes are sometimes followed by a more serious security event, but they are often the result of curiosity or confusion* (Dekker, 1997).

Protocol. *1. Convention or standard that controls or enables the connection, communication, and data transfer between two computing endpoints. In its simplest form, a protocol can be defined as the rules governing the syntax, semantics, and synchronization of communication* (`http://en.wikipedia.org/wiki/Protocol_(computing)`). *2. A set of rules (i.e., formats and procedures) to implement and control some type of association (e.g., communication) between systems* (Shirey, 2000).

Proxy. *Software process - often used as, or as part of, a firewall - that relays a protocol between client and server computer systems, by appearing to the client to be the server and appearing to the server to be the client* (Shirey, 2000).

Q

No entry.

R

Race Condition. *Undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence in order to be done correctly* (`http://searchstorage.techtarget.com/`).

Range. *How a vulnerability is explored* (CVE).

Remote Attack. *Attack that is initiated from outside the system, by an unauthorized or illegitimate user of the system* (Shirey, 2000).

Remote Execution. *Security incident that allows users on a client access to execute remote applications or codes.*

Repeater. *Device that receives a digital signal on an electromagnetic or optical transmission medium and regenerates the signal along the next leg of the medium* (`http://www.whatis.com`).

Resource. *Anything usable or consumable while performing a function in a system, such as: processors, memory, I/O devices, bandwidth, service and communication channel* (FC1, 1992).

Root Access. *Privileged access with full rights in a computer system.*

Root Account. *Special account with full rights in a computer system.* ***Synonymous***: *Administrator; Super User.*

Root Compromise. *Similar to an account compromise, except that the account that has been compromised has special privileges on the system. The term root is derived from an account on UNIX systems that typically has unlimited, or "super user", privileges. Intruders who succeed in a root compromise can do just about anything on the victim's system, including run their own programs, change how the system works, and hide traces of their intrusion* (Dekker, 1997).

Root ID. *Root Identifier.*

Root Toolkit. *1. Package with programs (scritp, malware) that allows an agent to perfom an attack and cause a security incident. 2. Software inserted onto a computer system after an attacker has gained control of the system. Rootkits often include functions to hide the traces of the attack, as by deleting log entries or cloaking the attacker's processes* (TrendMicro, 2006).

Router. *Computer or device that is a gateway between two networks at OSI (Open Systems Interconnection) layer 3 and that relays and directs data packets through that internetwork. The most common form of router operates on IP packets. A device or, in some cases, software in a computer, that determines the next network point to which a packet should be forwarded toward its destination. The router is connected to at least two networks and decides which way to send each information packet based on its current understanding of the state of the networks it is connected to* (Shirey, 2000).

S

Scam Phishing. *Scam in which someone sends out legitimate-looking emails appearing to come from some of the Web's biggest sites or organizations, in an effort to phish for personal and financial information from the recipient. Include hoax email messages that promise material gain or even luck to recipients who forward them to others users* (TrendMicro, 2006).

Scanner. *Computer program to probe a remote system, typically to discover vulnerabilities. Computer program designed to search an application, computer or network for weaknesses. The scanner systematically engages the target in an attempt to discover vulnerabilities* (`http://en.wikipedia.org/wiki/Vulnerability_scanner`).

Scanning. *1. Large number of probes done using an automated tool. Scans can sometimes be the result of a misconfiguration or other error, but they are often a prelude to a more directed attack on systems that the intruder has found to be vulnerable* (Dekker, 1997). *2. Procedure for identifying active hosts on a network, either for the purpose of attacking them or for network security assessment.* ***Synonymous***: *Network Scanning* (`http://www.whatis.com`).

Secure Shell (SSH). *Set of standards and an associated network protocol that allows establishing a secure channel between a local and a remote computer* (`http://en.wikipedia.org/wiki/Ssh`).

Secure Sockets Layer (SSL). *Cryptographic protocol which provides secure communications on the Internet* (`http://en.wikipedia.org/wiki/Transport_Layer_Security`).

Security Tool. *Program or application developed to protect computer systems.*

Security Incident. *Unauthorized access or violation of security policies* (CERT).

Self-Replicating. *Malware that can replicate themselves.*

Server Application. *System entity (software) that provides a service in response to requests from other system entities called clients* (Shirey, 2000).

Service. *Any non-material equivalent of a good provided to the user's systems* (`http://en.wikipedia.org/wiki/Service`).

Simple Mail Transfer Protocol (SMTP). *TCP-based, application-layer, Internet Standard protocol for moving electronic mail messages from one computer to another* (Shirey, 2000).

Simple Network Management Protocol (SNMP). *UDP-based, application-layer, Internet Standard protocol for conveying management information between managers and agents* (Shirey, 2000).

Sniffing. *Attack that intercepts and accesses data and other information contained in a flow in a communication system. It attempts to observe the flow and gain knowledge of information it contains.* ***Synonymous***: *Passive Wiretapping.* (Shirey, 2000)

Social Engineering. *Instead of collecting information by technical means, intruders might also apply methods of social engineering like impersonating individuals on the telephone, or using other persuasive means to encourage someone to disclose information* (West-Brown et al., 1998).

Software. *Computer programs (which are stored in and executed by computer hardware) and associated data (which also is stored in the hardware) that may be dynamically written or modified during execution* (Shirey, 2000).

Source Port. *Logical port from where a security incident had came.*

Spam. *1. Verb: To indiscriminately send unsolicited, unwanted, irrelevant, or inappropriate messages, especially commercial advertising in mass quantities. 2. Noun: electronic "junk mail". 3. Unsolicited email on the Internet* (Shirey, 2000).

Spoofing. *1. Attack in which one system entity illegitimately poses as assumes the identity of another entity. 2. It is the forgery of any message or packet header so that the message appears to have originated from someone or somewhere other than the actual source.* ***Synonymous***: *Masquerading* (Shirey, 2000).

Spyware. *Program that monitors and gathers user information for different purposes, and usually run in the background, with their activities transparent to most users* (TrendMicro, 2006).

SQL Injection. *Security incident that enables SQL arbitrary queries in the database layer of an application.*

SSH File Transfer Protocol (SFTP). *Network protocol designed by the IETF to provide secure file transfer and manipulation facilities over the Secure Shell Protocol (SSH)* (`http://en.wikipedia.org/wiki/SSH_file_transfer_protocol`).

Supplier. *Organization that develops or supplies hardware or/and software.*

Switch. *Device that channels incoming data from any of multiple input ports to the specific output port that will take the data toward its intended destination* (Shirey, 2000).

T

Telephone Network (Telnet). *Network protocol used on the Internet or local area network (LAN) connections. Typically used to provide user oriented command line login sessions between hosts on the Internet* (`http://en.wikipedia.org/wiki/Telnet`).

Theft of Resource. *Consequence implied when the agent steals resource from the system.*

Tool. *A means of exploiting a computer (host) or a computational system* (Howard, 1997).

Transmission Control Protocol (TCP). *A reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error on any other machine in the Internet. It fragments the incoming byte stream into discrete messages and passes each one onto the Internet layer* (Tanenbaum, 2003).

Transport Layer Security (TLS). *Cryptographic protocol which provides secure communications over the Internet* (`http://en.wikipedia.org/wiki/Transport_Layer_Security`).

Trojan Horse. *1. Computer program that appears to have a useful function, but also has a hidden and potentially malicious function that evades security mechanisms,*

sometimes by exploiting legitimate authorizations of a system entity that invokes the program (Shirey, 2000). *2. A normally trustworthy program or process modified to include unwanted and unknown functions that may (or can) compromise the security of the user, system, network, application, or protocol involved* (West-Brown et al., 1998).

U

Unauthorized User. *User who does not have specific right or permission to do something in a system.*

Uniform Resource Locator (URL). *String of characters conforming to a standardized format, which refers to a resource on the Internet (such as a document or an image) by its location* (Berners-Lee et al., 1998).

User Access. *Any regular access with limited rights in a computer system.*

User Account. *Any regular account with limited rights in a computer system.*

User Application. *Application that is no responsible to serve another application or users.*

User Datagram Protocol (UDP). *An unreliable, connectionless protocol for applications that do not want TCP's sequencing or flow control and wish to provide their own* (Tanenbaum, 2003).

User ID. *User Identifier.*

V

Virus. *Hidden, self-replicating section of computer software, usually malicious logic, that propagates by infecting or inserting a copy of itself into and becoming part of another program. A virus cannot run by itself; it requires that its host program be run to make the virus active* (Shirey, 2000).

Vulnerability. *1. Flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy* (Shirey, 2000). *2. Software weakness, design, or implementation error that can lead to an unexpected, undesirable event compromising the security of the system, network, application, or protocol involved* (West-Brown et al., 1998).

Vulnerability Type. *Defines the type of a vulnerability.*

W

Web Server. *A software process that runs on a host computer connected to the Internet to respond to HTTP requests for documents from client web browsers* (Shirey, 2000).

Website. *A collection of Web pages, typically common to a particular domain name or subdomain on the World Wide Web on the Internet* (`http://en.wikipedia.org/wiki/Website`).

Windows Internet Naming Service (WINS) Server. *Microsoft's implementation of NetBIOS Name Server (NBNS) on Windows, a name server and service for NetBIOS computer names* (`http://en.wikipedia.org/wiki/Windows_Internet_Naming_Service`).

Worm. *1. A computer program that can run independently, can propagate a complete working version of itself onto other hosts on a network, and may consume computer resources destructively* (Shirey, 2000). *2. A self-contained program (or set of programs) that is able to spread functional copies of itself or its segments to other computer systems. The propagation usually takes place via network connections or email attachments* (TrendMicro, 2006).

X

No entry.

Y

No entry.

Z

No entry.

A.2 Vocabulário de Relacionamentos

actson_Asset. *A Security Incident acts on an Asset.*

causes_SecurityIncident. *An Attack can cause a Security Incident.*

developsCorrection. *A Supplier develops a Vulnerability Correction.*

exploredwithVulnerability. *A Vulnerability is explored with other ones.*

explores_Vulnerability. *an Agent explores a Vulnerability using a Tool.*

gets_Access. *An Agent gets access to a computational system using a Tool.*

happenson_Asset. *A Vulnerability happens on an Asset.*

hasCorrection. *A Vulnerability has a Correction.*

hasCorrectionSupplier. *A Correction has Supplier.*

hasPrecondition. *A Logic Bomb needs PreCondition to be executed.*

hasPreCondition. *A Vulnerability has another Vulnerability as precondition.*

hasRange. *A Vulnerability has a Range.*

hasSupplier. *Software and Hardware have a Supplier.*

hasType. *A Vulnerability has a Type.*

hasVulnerability. *An Asset has a Vulnerability.*

has_Precondition. *A Security Incident needs a PreCondition to happen.*

impliesto_Consequence. *A Security Incident implies to a Consequence.*

isexploredby_Tool. *A Vulnerability is explored by a Tool.*

isusedby_SecurityIncident. *A Port is used during a Security Incident*

performsan_Attack. *An Agent performs an Attack to cause a Security Incident.*

precedesa_SecurityIncident. *A Security Incident can precedes another Security Incident.*

proceedsa_SecurityIncident. *A Security Incident can proceeds another Security Incident.*

preconditionto. *A Vulnerability is a precondition to another Vulnerability.*

relatesto_Asset. *A Consequence is related to an Asset.*

relatesto_Backdoor. *A Trojan Horse is related to a Backdoor.*

relatesto_Spyware. *A Trojan Horse is related to a Spyware.*

relatesto_TrojanHorse. *A Backdoor is related to a Trojan Horse.*

relatesto_THorse. *A Spyware is related to a Trojan Horse.*

relatesto_Vulnerability. *A PreCondition is related to a Vulnerability.*

sellsProduct. *A Supplier supplies or develops Hardware and Software.*

usedby_SecurityIncident. *A Protocol is used during a Security Incident*

usesPort. *A Software uses a Port.*

usesProtocol. *A Software uses a Protocol.*

uses_Port. *A Security Incident uses a Protocol.*

uses_Protocol. *A Security Incident uses a Protocol.*

uses_Tool. *An Agent uses a Tool to cause a Security Incident.*

Apêndice

B

Boletins sobre a vulnerabilidade "CVE-2005-1983"

Boletim CVE

```
CVE-2005-1983, Candidate,
"Stack-based buffer overflow in the Plug and Play (PnP) service for Microsoft Windows
2000 and Windows XP Service Pack 1 allows remote attackers to execute arbitrary code
via a crafted packet, and local users to gain privileges via a malicious application,
as exploited by the Zotob (aka Mytob) worm.", "MS:MS05-039"
| URL:http://www.microsoft.com/technet/Security/bulletin/ms05-039.mspx
| ISS:20050809 Windows Plug and Play Remote Compromise
| URL:http://xforce.iss.net/xforce/alerts/id/202
| CERT:TA05-221A
| URL:http://www.us-cert.gov/cas/techalerts/TA05-221A.html
| CERT-VN:VU#998653
| URL:http://www.kb.cert.org/vuls/id/998653
| CIAC:P-266
| URL:http://www.ciac.org/ciac/bulletins/p-266.shtml
| MISC:http://www.hsc.fr/ressources/presentations/null_sessions/
| MISC:http://www.securiteam.com/windowsntfocus/5YP0E00GKW.html
| MISC:http://www.frsirt.com/english/alerts/20050814.ZotobA.php
| FULLDISC:20050811 Windows 2000 universal exploit for MS05-039
| URL:http://archives.neohapsis.com/archives/fulldisclosure/2005-08/0384.html
| BID:14513
| URL:http://www.securityfocus.com/bid/14513
| FRSIRT:ADV-2005-1354
| URL:http://www.frsirt.com/english/advisories/2005/1354
| OVAL:OVAL100073
| URL:http://oval.mitre.org/oval/definitions/data/oval100073.html
| SECUNIA:16372
| URL:http://secunia.com/advisories/16372
```

| SECTRACK:1014640
| URL:http://securitytracker.com/id?1014640
| OSVDB:18605
| URL:http://www.osvdb.org/18605
| XF:win-plugandplay-bo(21602)
| URL:http://xforce.iss.net/xforce/xfdb/21602,Assigned (20050617)

Parte do Boletim da Microsoft

FONTE: http://www.microsoft.com/brasil/security/boletins/ms05-039.mspx.

Boletim de Segurança da Microsoft MS05-039
A vulnerabilidade no Plug and Play pode permitir a execução remota de código
e a elevação de privilégio (899588)

Publicado em: 9 de Agosto de 2005
Versão: 1.0

Resumo
Quem deve ler este documento: clientes que usam o Microsoft Windows
Impacto da vulnerabilidade: Execução remota de código e elevação local de privilégio
Classificação máxima de gravidade: Crítica
Recomendação: Os clientes devem aplicar a atualização imediatamente.
Substituição da atualização de segurança: Nenhum
Advertências: Nenhuma
Locais de software testado e de download de atualização de segurança:
Software afetado:
Microsoft Windows 2000 Service Pack 4
Microsoft Windows XP Service Pack 1 e Microsoft Windows XP Service Pack 2

FONTE: http://www.microsoft.com/brasil/security/advisory/899588.mspx.

Comunicado de Segurança da Microsoft (899588)
A vulnerabilidade no Plug and Play pode permitir a execução remota de código
e a elevação de privilégio

Publicado em: 11 de Agosto de 2005 | Atualizado em: 17 de Agosto de 2005

A Microsoft está analisando ativamente e fornecendo orientação sobre um worm mal-intencionado
que circula atualmente na Internet, bem como suas variantes, e é identificado como
"Worm:Win32/Zotob.A". O worm é um ataque mal-intencionado que explora a vulnerabilidade do
Plug and Play do Windows mencionada no Boletim de Segurança da Microsoft MS05-039 de 9 de
agosto de 2005.

Nossas investigações iniciais revelaram que o worm ataca sistemas baseados no Windows 2000.
Para obter mais informações sobre este worm, para ajudar a determinar se seu computador foi
infectado por ele e para obter instruções sobre como reparar o sistema no caso de um ataque
por este worm, consulte o site do Zotob Security Incident ou a Microsoft Virus Encyclopedia
(Worm:Win32/Zotob.A, Worm:Win32/Zotob.B).

Outras versões do Windows, incluindo o Windows XP Service Pack 2 e o Windows Server 2003,
não serão afetadas pelo "Worm:Win32/Zotob.A", a menos que já estejam comprometidas por outros
softwares mal-intencionados. Os clientes podem proteger seus computadores de ataques instalando

imediatamente as atualizações de segurança fornecidas pelo Boletim de Segurança da Microsoft MS05-039.

A Microsoft está investigando este ato mal-intencionado para continuar a entender como oferecer suporte aos seus clientes. A Microsoft está ciente de que algumas versões detalhadas do código de exploração foram publicadas na Internet para a vulnerabilidade citada no Boletim de Segurança da Microsoft MS05-039: A vulnerabilidade no Plug and Play pode permitir a execução remota de código e a elevação de privilégio (899588).