HUNG RUO HAN

Proposta de descrição de ataques em ambientes grid

Florianópolis, 2007

HUNG RUO HAN

Proposta de descrição de ataques em ambientes grid

Projeto de Pesquisa para elaboração do Trabalho de Conclusão de Curso apresentado como exigência para a obtenção do título de Bacharel em Ciências da Computação à Universidade Federal de Santa Catarina – UFSC, no curso de Ciências da Computação.

Orientador: Prof. Carlos Becker Westphall

Florianópolis, 2007

UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

A COMISSÃO EXAMINADORA, ABAIXO ASSINADA, APROVA O TRABALHO DE CONCLUSÃO DE CURSO

Proposta de descrição de ataques em ambientes grid

ELABORADO POR

HUNG RUO HAN

COMO REQUISITO PARCIAL PARA A OBTENÇÃO DO TÍTULO DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

COMISSÃO EXAMINADORA:

_____

Carla Merkle Westphall, Dr.

_____

Fernando Luiz Koch, Dr.

_____

Kleber Magno Maciel Vieira, MSc.

Florianópolis, ___ de outubro de 2007.

Para as pessoas especiais que me ensinam como ser uma pessoa melhor a cada dia.

Minha Família

RESUMO

Com o crescente esforço para incrementar a segurança em sistemas de computação em *grid*, técnicas de detecção de intrusão vêm sendo sugerida pela literatura. Este trabalho tem por objetivo padronizar a descrição de ataques de intrusões nesse tipo de ambiente, pois até agora as linguagens conhecidas que abordam o assunto possuem uma documentação ruim (devido ao abandono de projetos) ou não são viáveis para um ambiente de computação em *grid*. A proposta deste trabalho é padronizar uma descrição que defina um ataque num ambientes de computação em *grid*. Com essa padronização, será mais fácil para os administradores de ambientes em *grid* compartilharem informações, tornando esse tipo de ambiente cada vez mais seguro, e por conseqüência, tendo mais usuários querendo utilizar essa tecnologia.

Palavras-chave: Computação em *grid*. Sistema de Detecção de Intrusão. Especificação de intrusão.

ABSTRACT

The specialized literature suggests that techniques for grid computing intrusion detection are a solution to improve the security in these environments. However, the lack of a standard language to describe intrusion is a shortcoming in current works. This work aims to provide a standard definition. We argue this standardization will ease the administration efforts, promote knowledge sharing and, ultimately, help with the technology's dissemination.

Key-words: Grid computing. Intrusion Detection System. Intrusion specification.

LISTA DE SIGLAS

ASCCI – American Standard Code for Information Interchange

CISL – Common Intrusion Specification Language

CIDF – Common Intrusion Detection Framework

CORBA – Common Object Request Broker Architecture

DCOM – Distributed Component Object Model

DNS – Domain Named Service

DTD – Document Type Definition

GIDS – Grid-based Intrusion Detection System

HTML – Hyper Text Markup Language

IDS – Intrusion Detection System

NIDS – Network Intrusion Detection System

RMI – Remote Method Invocation

RPC – Remote Procedure Call

W3C – World Wide Web Consortium

XML – extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Hoje em dia a internet se tornou um meio de comunicação amplamente utilizado. O próximo passo na evolução da internet seria integrar vários computadores entre si com o objetivo de aproveitar ao máximo o desempenho e serviços disponíveis por eles.

A nomenclatura *grid* (*Grid Computing*) é baseada nas *malhas* de interligação dos sistemas de energia elétrica. Em um sistema elétrico, o usuário utiliza a energia sem se perguntar aonde a mesma foi gerada. De maneira semelhante, a idéia de um *grid computacional* é que possamos criar um ambiente computacional distribuído que possua mecanismos que permitam o processamento, o armazenamento e o uso de equipamentos de forma transparente para os usuários da configuração [DANTAS].

A idéia deste trabalho é fazer um estudo sobre ambientes em *grid*, mais especificamente sobre o sistema de detecção de intrusão.

1.1 OBJETIVO GERAL

A proposta do trabalho de conclusão de curso é propor as regras que definem um intruso num ambiente grid, através de uma linguagem escolhida. Contribuindo assim, para a melhora do Sistema de Detecção de Intrusão (IDS – *Intrusion Detection System*). A idéia é que essas regras sejam bastante utilizadas em diversas plataformas *grid*, tornando assim a linguagem e as regras padrão para especificação de intrusos nesse tipo de ambiente.

Com essa possível padronização administradores e consultores não precisariam estar aprendendo uma linguagem nova de especificação de regras de intrusão cada vez que eles entrarem em contato com uma nova plataforma *grid*.

Além disso, é possível que administradores de diferentes plataformas *grid* consigam trocar experiências, fazendo com que todos consigam usar uma nomenclatura em comum. Assim os sistemas de ambiente em *grid* tornam-se cada vez mais seguros, beneficiando os usuários finais.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desse trabalho são:

- Obter conhecimentos sobre IDS;
- Obter conhecimentos sobre as linguagens de especificação de regras;
- Definir a arquitetura do XADG (eXtensible Attack Definition in Grid);
- Implementar um serviço que disponibiliza uma definição de ataque atualizada para o IDS.

1.3 MOTIVAÇÃO

Para que a computação em *grid* se torne mais comercial, é necessário uma plataforma mais estável e seguro. Assim, a quantidade de usuários que irão utilizá-la aumentarão significativamente.

Como as questões de segurança na área da computação são muito complexo (devido a má intenção, falta de informação, inexperiência de alguns usuários e outros motivos) fica difícil conseguir fazer com que os usuários se sintam seguros ao utilizar os serviços computacionais, por exemplo, hoje em dia muitos usuários ainda não se sentem seguros em pagar as suas contas via internet.

A partir dessa motivação, o trabalho tem por objetivo dar uma contribuição para a melhoria da segurança da computação *grid*. Por conseqüência, contribuindo para a evolução da computação nesse tipo de ambiente. Pois quanto mais um usuário se sente seguro ao utilizar o sistema, mais ele vai utilizar esse sistema. O

usuário gostando de utilizar o sistema é possível que ele divulgará para os seus amigos e colegas de trabalhos, esses por sua vez, falaram para os próximos amigos. Conseqüentemente, aumentará a quantidade de usuários que utilizam esse sistema.

O texto está organizado em 6 capítulos: o capítulo 2 aborda os fundamentos teóricos necessários para tratar do assunto deste trabalho; o capítulo 3 descreve a proposta da linguagem de especificação de intrusão; o capítulo 4 apresenta os resultados obtidos da aplicação desenvolvida; o capítulo 5 apresenta a conclusão do trabalho, e por fim, o capítulo 6 apresenta os trabalhos futuros.

2 FUNDAMENTOS TEÓRICOS

Este capítulo possui breves explicações sobre os assuntos relevantes para o entendimento do trabalho. Ele está organizado da seguinte maneira: a seção 2.1 aborda sobre a computação em *grid*; 2.2 aborda sobre o IDS; 2.3 aborda as linguagens de especificações de intrusão conhecidas; 2.4 aborda sobre a linguagem XML (eXtensible Markup Language).

2.1 COMPUTAÇÃO EM GRID

Atualmente utilizamos muito os computadores para nos ajudar com as tarefas do dia a dia. Mas sempre existem problemas que os computadores convencionais não são capazes de solucionar, sejam eles biológicos, médicos, meteorológicos, matemáticos, entre outros. Para tanto, surgiram os supercomputadores, máquinas extremamente caras, específicas para aplicações científicas que requerem altíssimo desempenho.

Segundo a definição [TANENBAUM] um sistema distribuído é uma “coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente”. Sendo assim, a computação distribuída consiste em adicionar o poder computacional de diversos computadores interligados por uma rede de computadores trabalhando em conjunto no mesmo computador, para processar colaborativamente uma determinada tarefa de forma coerente e transparente, ou seja, trabalhando como se fosse apenas um único e centralizado computador.

A união desses diversos computadores com o objetivo de compartilhar a execução de tarefas é chamado de sistema distribuído. Exemplos de computação distribuída são: CORBA (*Common Object Request Broker Architecture*), RPC

(*Remote Procedure Call*), RMI (*Remote Method Invocation*), DCOM (*Distributed Component Object Model*), agente móveis, *peer to peer* e *Grid*.

A computação em *grid* tem por objetivo fornecer uma visão transparente do sistema, como se o conjunto de *hardware* e *software* (que forma a *grid* computacional) fosse um único e poderoso computador. Além disso, a computação em *grid* possibilita a integração de redes de diferentes instituições de forma dinâmica, para compartilhamento de ciclos de processador, espaço em disco, acesso a banco de dados, *software*, arquivos, ou qualquer outro tipo de recurso disponível. A figura 2.1 mostra a evolução das tecnologias utilizadas em redes computacionais.

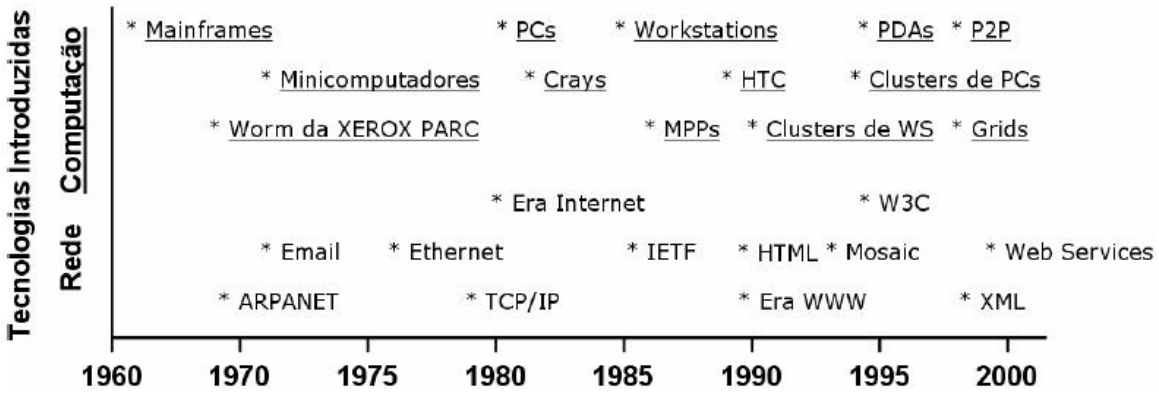


FIGURA 2.1 TECNOLOGIAS UTILIZADAS EM REDES COMPUTACIONAIS.

É desejável que quando um usuário se encontra num ambiente em *grid*, ele apenas se preocupe em fazer a requisição para o ambiente. Esse ambiente que o usuário se encontra se encarregaria de distribuir a tarefa para os outros nodos do ambiente. Fazendo uma analogia ao sistema de energia elétrica, quando um usuário liga o seu aparelho eletrônico na tomada de casa, ele não sabe de onde está vindo aquela energia.

A figura 2.2 ilustra a idéia da computação em *grid* como um exemplo. Onde um usuário de um nodo faz uma requisição de um certo serviço para o ambiente e

este se encarrega de encontrar o(s) nodo(s) que possuem o serviço requisitado e estabelecer as conexões entre os nodos.

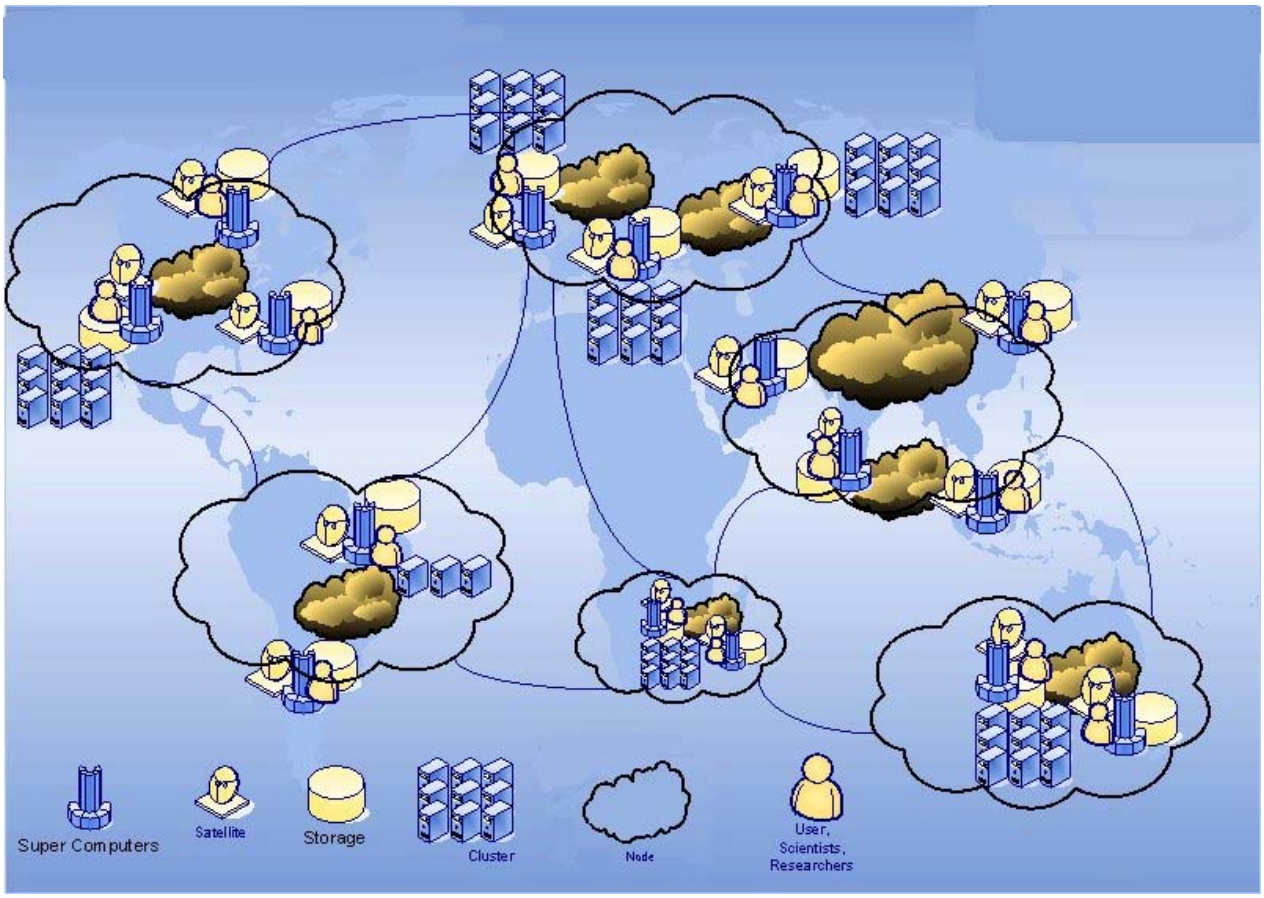


FIGURA 2.2 UM AMBIENTE EM *GRID*.

2.1.1 VANTAGENS

A computação em *grid* é considerada uma evolução natural da computação distribuída, cujos benefícios são levados ao extremo. Com o aumento da dispersão de um conjunto de dados de uma organização, aumenta também sua disponibilidade e velocidade de aquisição, já que o link de um servidor dificilmente ficará sobrecarregado, além de resistir a quedas e tentativas de ataques de negação de serviço.

Uma boa vantagem sobre as outras tecnologias é o reaproveitamento das máquinas, evitando gastos excessivos para as organizações. O fuso horário é um

aliado da computação em *grid*, pois enquanto estamos em horário comercial aqui no Brasil, a maioria dos computadores dos países como a China estão ociosos. Com essa tecnologia é possível utilizar esses computadores ociosos, ao invés de comprar computadores mais poderosos para atender a demanda das organizações.

Outro benefício que pode ser obtido através da computação em *grid* é o compartilhamento de dados entre organizações de pesquisa, além do próprio compartilhamento de ciclos de processamento. As organizações de pesquisa geralmente precisam de um enorme poder de processamento para realizar simulações ou análises de dados. Um sistema *cluster* resolveria o problema, porém utilizando a computação em *grid* não há necessidade de se utilizar computadores dedicados ao sistema, além de não ser preciso que eles estejam na mesma sala, cidade, estado ou país.

2.2 SISTEMAS DE DETECÇÃO DE INTRUSÃO

Um IDS (*Intrusion Detection System*) é um sistema que detecta, de preferência em tempo real, violações de segurança em um sistema de informação e envia notificações de alerta ao gerente de segurança (pode ser tanto um ser humano ou uma máquina), o qual tem poder e direito de tomar as medidas mais apropriadas, como desconectar um usuário ou bloquear o tráfego de rede.

Um IDS não toma medidas preventivas quando um ataque é detectado, eles fazem o papel de informantes e não o policiais. O IDS é similar ao sistema de alarmes de ladrões, os mesmos problemas ocorrem em ambos os sistemas, como situações de alarme falso e intrusos que enganam o sistema. Mas o sistema de alarme de ladrões é mais simples porque normalmente o sistema só é ativado quando não há atividade considerado normal, ou seja, quando o expediente já foi

encerrado ou quando todos de uma residência já foram dormir. Assim, qualquer atividade que o sistema conseguir encontrar é considerado como um intruso.

No IDS a situação é diferente porque sempre está tendo atividades no sistema, não é viável e nada prático parar o sistema todo para poder detectar um intruso. Ainda mais na computação em *grid* que a intenção é que os computadores sejam utilizado na maior parte do tempo possível, por exemplo, os computadores do Brasil a noite seriam muito úteis para ajudar no processamento de dados ou qualquer outro tipo de serviço computacional requisitado na China.

2.2.1 TIPOS DE IDS

Para podermos detectar uma intrusão acreditamos que o comportamento de um intruso será diferente de um usuário legítimo e de que muitas ações não-autorizadas serão detectadas. As técnicas para identificarmos os intrusos são geralmente classificadas em técnicas baseadas em anomalias e técnicas baseadas em assinaturas, cada um com suas vantagens e desvantagens.

Sistemas baseados em anomalias (comportamentos) aprendem o comportamento normal ou esperado dos usuários que utilizam o sistema e tentam identificar as divergências de uso. Se o uso está fora de um determinado limiar, uma notificação é disparada ao gerente de segurança.

Sistemas baseados em assinaturas (regras) ficam a procura de rastros ou padrões conhecidos resultantes do uso malicioso do sistema que ficam armazenados em um banco de dados de assinaturas de ataques. Dependendo da definição dessas assinaturas, esses sistemas podem até detectar ataques desconhecidos que são similares a alguns ataques conhecidos.

2.3 DETECÇÃO DE INTRUSÃO BASEADO EM REGRAS

Essa técnica é a mais utilizada nos Sistemas de Detecção de Intrusão porque possui um baixo índice de alarmes falsos e um elevado índice de acertos. A limitação dessa técnica é que raramente consegue detectar padrões de ataques desconhecidos. Essa técnica é baseada em regras que monitoram um fluxo de eventos e agem como um algoritmo que procura as características maliciosas em um comportamento.

Utilizando um sistema especialista é possível descrever o comportamento malicioso em uma regra. As regras facilitam a evolução dos sistemas de detecção de intrusão porque uma nova regra pode ser adicionada sem mudar as regras existentes. Diferente dos sistemas baseados em anomalias (comportamentos) que toda vez que acrescenta uma nova característica é necessário realizar todo o aprendizado novamente (treinamento da rede neural).

Como é possível observar a produção de regras é o elemento chave desta técnica, pois é através dela que o sistema consegue reconhecer um ataque no ambiente. Elaborando um conjunto de regras boas pode assegurar uma boa segurança ao sistema. Vale lembrar que regras em excesso acabam restringindo a liberdade do usuário e acabam tornando o sistema inviável de ser utilizado.

Nas próximas subseções algumas linguagens de especificação de intrusão serão apresentadas.

2.3.1 CISL

O CISL (*Common Intrusion Specification Language*) foi desenvolvido a partir do CIDF (*Common Intrusion Detection Framework*). O CIDF tem por objetivo desenvolver protocolos e interfaces das aplicações para que os projetos na área de

detecção de intrusão possam compartilhar informações e recursos. Tornando possível a reutilização dos componentes de detecção de intrusão em outros sistemas.

O número de pessoas ou máquinas que se passam por intrusos nos sistemas dos dias de hoje estão crescendo em grande escala. Em cada ambiente, a habilidade do IDS e o compartilhamento de informações com os seus componentes são muito importantes. Se um novo sistema possuir as informações de como os outros sistemas foram atacados, a probabilidade de que esse novo sistema seja atacado da mesma maneira diminui bastante, favorecendo os administradores de sistemas.

O CISL foi criado para poder realizar essas trocas de informações, pois é necessário que os sistemas utilizem a mesma linguagem para que todos se entendam. Segundo a documentação [FEIERTAG], uma linguagem desenvolvida para troca de informações sobre ataques deve possuir as seguintes qualidades:

1. Expressivo. Componentes devem ser capaz de expressar bastante variedades dos intrusos, com pouco texto e numa ordem padrão.
2. Unicidade nas expressões. Não deve ter várias maneiras para expressar a mesma expressão.
3. Preciso. O significado de uma expressão utilizando a linguagem deve ser bem definida.
4. Divisível em camadas. Significados específicos devem ser expressos de uma forma generalizada, para que destinatários com requisições diferentes possam entender as mensagens.

5. Bem definido. As mensagens tem que ser facilmente interpretadas, evitando que o emissor da mensagem tenha que explicar o que ele está querendo dizer.
6. Eficiente. A mensagem deve consumir o mínimo de recursos do sistema, evitando que informações similares se repitam.
7. Extensível. Se for necessário adicionar informações nas expressões, tem que ser feito com o mínimo de esforço e não de forma a perder compatibilidade com outros componentes CIDF que não conheça as suas extensões.
8. Simples. As pessoas consigam passar as informações rapidamente. Isso facilita para quem for receber as informações não tenham muito trabalho para entender as mensagens também.
9. Portátil. A linguagem deve suportar a variedade de plataformas e mecanismos de transporte.
10. Fácil implementação. Se a linguagem for muito difícil de se implementar, ninguém irá utilizar.

Tendo uma linguagem que torne essas características possíveis, sendo seguro, eficiente, entre outros. A transmissão das informações poderão garantir características mais confiáveis como autenticação, prioridade das mensagens trocadas e etc.

A linguagem proposta no texto possui a sintaxe bastante similar a língua Inglesa. Foi utilizado *S-expressions*[1] que é um tipo de estrutura de dados para representar dados mais complexos; geralmente, o agrupamento é feito por

[1] Maiores informações em: http://theory.lcs.mit.edu/~rivest/sexp.html

parênteses, assim como em LISP. A figura 2.3.1.1 é um exemplo de uma simples expressão-S:

```
(HostName 'ten.ada.net')
```

FIGURA 2.3.1.1 EXEMPLO DE EXPRESSÃO-S.

A vantagem de expressões-S é que eles determinam que os termos sejam explicitamente associados, sem limitar o que aqueles termos e os seus grupos devem expressar.

Para demonstrar melhor o CISL, será mostrada a figura 2.3.1.2:

```
(Delete
        (When
                (Time '12:24 15 Mar 1999 UTC')
        )
        (Initiator
                (UserName 'joe')
                (UserID 1234)
                (HostName 'foo.example.com')
        )
        (FileSource
                (FullPathName '/etc/passwd')
                (HostName 'foo.example.com')
        )
)
```

FIGURA 2.3.1.2 EXEMPLO DE CÓDIGO CISL.

A expressão que representa a ocorrência é chamada de **sentença**, é análoga a sentenças da língua Inglesa. Assim como as sentenças da língua Inglesa, as sentenças CISL possuem sujeitos, verbos e objetos diretos. Outra semelhança entre as duas linguagens é que o verbo é considerado o coração da sentença, o restante da sentença é interpretado de acordo com o verbo. A posição e a organização das sentenças CISL refletem nas suas prioridades.

Na sentença da figura 2.3.1.2, o verbo é *Delete*. É utilizado para expressar o fato que alguém deletou ou tentou deletar um arquivo. Para identificar a pessoa que andou deletando o arquivo, e qual o arquivo que está sendo deletado, a sentença inclui algumas cláusulas. Essa cláusulas servem de identificadores das sentenças que são interpretadas de acordo com o verbo. No exemplo acima, *Initiator* identifica a entidade que realiza o ato de deletar e *FileSource* identifica o arquivo que está sendo deletado.

Adicionalmente, o CISL também inclui facilidades para modificar o verbo por exemplo, dizendo quando e onde o evento ocorreu. Essa informação é colocado nos advérbios. No exemplo, *when* é o advérbio dizendo quando o ato ocorreu.

Agora temos informações suficientes para produzir a interpretação completa da sentença da figura 2.3.1.2:

No dia 15 de março de 1999, as 12:24 no horário universal. O usuário ‘joe’ que possui o UID 1234 deletou o arquivo /etc/passwd na máquina chamada ‘foo.example.com’.

2.3.1.1 PARADIGMAS DE ATAQUE

Um paradigma importante para ser abordado é o paradigma em relação a ataques que se divide em três aspectos:

- **saída**: O que o ataque atingiu? Por exemplo, o final de um resultado pode ser a aquisição de privilégios de *root* de uma máquina.
- **Classe do ataque**: Quais as formas convencionais de se praticar um ataque?
- **Mecanismo**: Quais foram as ações específicas realizadas?

Nem sempre acontecerá os três aspectos. Se um ataque falhar, por exemplo, não haverá a **saída**: nenhuma alteração significativa no sistema foi feita. Apesar do ataque ter falhado será notificado a **classe do ataque**.

É importante lembrar que esses três eventos não podem ser vistos separadamente, o ideal é que elas sejam vistas como três visões diferentes de um mesmo evento.

Suponha o seguinte ataque: *fingerd attack*[2]. Suponha também que o ataque seja bem sucedido e que a pessoa que realiza o ataque consiga privilégio de *root*[3]. A figura 2.3.1.1.1 mostra um exemplo da linguagem juntando as três ações e analisando-a como um evento só. Foi separado devido a mais fácil visualização e explicação, mas considere que o que está abaixo em <resultado>, <classe de ataque> e <mecanismo> estejam tudo dentro de ByMeansOf.

(ByMeansOf

<resultado>

<classe de ataque>

<mecanismo>

)

FIGURA 2.3.1.1.1 EXEMPLO UTILIZANDO O BYMEANSOF

[2] *fingerf attack*: realização de *overflow* no *buffer*.
[3] *Root*: privilégios totais no sistema (administrador).

Como o resultado da ação foi com que obteve acesso no endereço ´small.world.com´ e privilégios de *root*. Então, o <resultado> ficaria assim:

```
(AcquireProxy
        (World Unix)
                (When
                        (Time 15:34:10 27 Mar 1999 UTC)
                )
        (Initiator
                (HostName 'big.evil.com')
        )
        (Proxy
                (HostName 'small.world.com')
                (UserName 'root')
        )
)
```

Dependendo dos recursos de cada máquina, assim como até quais máquinas é possível acessar a partir daquela inicial. Vamos supor que nesse caso, ao obter privilégio de *root* a gravidade seja de 50 (numa escala de 0 a 100). Então a <classe de ataque> ficaria:

```
(Attack
        (Outcome
                (ReturnCode success)
        )
        (When
                (Time 15:34:10 27 Mar 1999 UTC)
        )
```

```
        (Initiator
                (HostName 'big.evil.com')
        )
        (Target
                (HostName 'small.world.com')
        )
        (AttackSpecifics
                (AttackID 0x0000000100000057)
                (Certainty 100)
                (Severity 50)
        )
)
```

Supondo que a forma do ataque foi uma mensagem na porta 79 com um vírus que possui grande capacidade de destruição. Isso causaria dois sub-eventos, conexão na porta 79 e envio da mensagem. O <mecanismo> ficaria:

```
(And
        (ConnectTCP
                (When
                        (Time 15:34:05 27 Mar 1999 UTC)
                )
                (Initiator
                        (HostName 'big.evil.com')
                        (ReferAs 0x01010101)
                )
                (Receiver
                        (HostName 'small.world.com')
                        (StandardTCPPort 79)
```

```
                (ReferAs 0x02020202)
            )
        )
        (SendMessage
            (When
                (Time 15:34:10 27 Mar 1999 UTC)
            )
            (Initiator
                (ReferTo 0x01010101)
            )
            (Receiver
                (ReferTo 0x02020202)
            )
            (Message
                (ByteSize 72348)
            )
        )
)
```

2.3.1.2 LIMITAÇÕES DO CISL

- Codificação muito limitada, só é possível descrever ataques de tamanho reduzido e com pouco tempo de duração;
- Muito antigo, documentação muito fraca. A documentação está na versão rascunho, a última atualização foi feita em 1999 e não possui uma lista dos verbos que a linguagem suporta. Provavelmente o projeto está parado;
- O CISL praticamente não se preocupa com as contas dos usuários;
- O CISL oferece as junções das expressões através do “ByMeansOf”, mas não é verificado se as expressões fazem sentido entre si. Um intruso podia ficar mandando mais expressões e ocupando o IDS de verifica-los. Por exemplo: o

IDS recebe primeiro uma expressão que alguém se *logou* no sistema; um intruso pode colocar várias expressões do tipo mensagem na porta 80 (muito utilizado pelo protocolo http em conversações web), ocupando o IDS com essas verificações para que ele realize outros serviços.

- Linguagem muito fácil de se confundir devido a quantidade elevada de parênteses, como dito anteriormente, muito parecido com LISP.

2.3.2 SNORT

O *SNORT* é uma ferramenta *NIDS* (Network Intrusion Detection System – "Sistemas de detecção de Invasão de Redes") que detecta intrusão numa rede e é capaz de prevenir o sistema de ataques através de análise de tráfego na rede nos *IP*[4]s da rede. O *SNORT* monitora o tráfego de pacotes em redes IP, realizando análises em tempo real sobre diversos protocolos (nível de rede e aplicação) e sobre o conteúdo (hexadecimal e ASCII[5]).

Existe também, a possibilidade de utilizar métodos como o *Database Plug-in*[6] por exemplo, para registrar pacotes em uma variedade de bases de dados diferentes (MySQL, PostgreSQL, entre outros), as quais contam com recursos próprios para efetuar consultas, correlações e dispõem de mecanismos de visualização para analisar dados.

Outro ponto positivo desse software é o grande número de possibilidades de tratamento dos alertas gerados. O subsistema de registro e alerta é selecionado em tempo de execução através de argumentos na linha de comando. Esses tratamentos são configurados através de regras que definem quando o sistema recebe um alerta,

---

[4] IP – Internet Protocol, protocolo utilizado entre duas máquinas em rede para encaminhamento dos dados.
[5] ASCII – American Standard Code for Information Interchange
[6] plug-in é geralmente um programa pequeno que serve para adicionar funções a outros programas maiores, provendo alguma funcionalidade especial ou específica, nesse caso um banco de dados.

quais informações devem ser mantido em um *log*[7], quais informações devem ser desconsiderados e etc.

Segundo a documentação do *SNORT*, essas regras são descritas através de uma linguagem simples, flexível e bastante poderosa. A maioria das regras do SNORT são escritas em uma simples linha.

2.3.2.1 REGRAS DO SNORT

As regras são divididas em duas sessões lógicas, o cabeçalho da regra e as opções da regra. O cabeçalho contem a ação da regra, protocolo, fonte, destino, endereço IP, máscara de rede, as portas de saída e entrada. As opções contem mensagens de alertas e informações em quais partes do pacote devem ser verificadas para determinar se a ação dita na regra deve ser ativada.

Abaixo um exemplo de uma regra do snort:

alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|"; msg:"mountd access";)

O texto até o primeiro parêntese é a parte considerada cabeçalho e o que está entre os parênteses são as opções da regra. As palavras que vem antes dos *dois pontos*, por exemplo o *content* e o *msg* são as palavras opcionais. Note que as opções da regra não são especificadas em nenhuma regra, eles são utilizados apenas para fazer as definições de pacotes e fazer os alertas. Todos os elementos que constituem a regra deve ser verdadeiro para indicar que a ação da regra está sendo tomada. Quando tomados juntos, os elementos podem ser consideradas na forma lógica.

---

[7] log – registro de eventos no sistema.

2.3.2.2 CABEÇALHO DA REGRA

No cabeçalho de uma regra contem informações que definem de quem é, de onde vem e o que é o pacote. O primeiro elemento é a ação da regra, a ação define o que fazer quando encontra o pacote que atende os critérios da regra. As três ações possíveis são:

- *Alert* – gera um aviso utilizando o método selecionado e então registra as ações do pacote;
- *Log* – registra as ações do pacote;
- *Pass* – ignora o pacote.

O próximo campo do cabeçalho é o protocolo. Os três protocolos que o *SNORT* acredita que tem comportamentos suspeitos são: tcp, udp e icmp. Futuramente será abordado outros protocolos como: ARP, IGRP, GRE, OSPF, RIP, IPX e etc.

Próximo campo do cabeçalho são o endereço IP e a porta. A palavra *any* pode ser usado para definir qualquer endereço. O *SNORT* não possui um mecanismo para fornecer o *hostname*(busca em DNS[8]), é necessário que passe o endereço IP.

Ainda tem a direção do operador “->” que indica pra que direção que se aplica a regra. O endereço IP e o número da porta no lado esquerdo da operação de direção indica que é a fonte e o endereço IP e a porta do lado direito é o destino. Existe também o operador que representa o bidirecional, que é indicado pelo símbolo “<>”.

---

[8] DNS – Domain Named Service

2.3.2.3 OPÇÕES DA REGRA

As opções da regra formam o coração do mecanismo de detecção de intrusão do *SNORT*, combinando-os da forma correta torna a linguagem poderosa e flexível. Todas as opções no *SNORT* são separados pelo “;”. As opções e os seus argumentos são separados através do “:”. Abaixo segue uma lista de algumas opções desses regras:

- Msg
- Logto
- Minfrag
- Dsize
- Content
- Flags
- Seq
- Ack
- Session

2.3.2.4 AQUITETURA DO SNORT

A implementação do *SNORT* focaliza a otimização do desempenho na coleta e análise de pacotes, que com o conjunto de regras correto aproxima-se do tempo real. O *SNORT* oferece o conceito de plug-in, que são seus subsistemas:

- Pré-processamento: Fica entre o pacote *Sniffing*[9] e o processamento do mecanismo de detecção, responsável por decodificar o pacote.
- Detecção: processamento do mecanismo de detecção

[9] Sniffing é realizado por uma ferramenta conhecida como Sniffer que é capaz de interceptar e registrar o tráfego de dados em uma rede de computadores.

- Saída: executado após o mecanismo de detecção, para registrar e alertar.

Primeiro, o decodificador de pacote interpreta qual é o conjunto de tráfego. Esse tráfego decodificado é analisado pelo *plug-in* pré-processador definido nas regras. Em seguida, o mecanismo de detecção recebe o tráfego, aplica uma estrutura de regras e pode também aplicar *plug-in* de detecção no tráfego excessivo para procurar outras assinaturas. Por fim, o fluxo de dados é enviado para o estágio de saída, onde existem diferentes opções de registro e alerta, e ainda, podem ser enviados para *plug-in* de saída que atuam como extremidade final de processamento para detecção.

2.3.2.5 CARACTERÍSTICAS DO SNORT

- O Snort rastreia um determinado faixa de IP. Num ambiente *grid* isso não seria viável porque nem sempre eles estão na mesma rede ou sub-rede;
- As regras do snort são em relação aos IPs. Num ambiente *grid* não é viável fazer o controle das ações por IP e sim por *login* de usuário.
- Documentação muito boa, existe uma comunidade que compartilha as regras do *SNORT*. Pena que não é viável para o ambiente distribuído porque é um IDS que analisa tráfego de rede (NIDS). O trabalho [SCHULTER] explica melhor as diferenças entre NIDS e GIDS (Grid-based Intrusion Detection System).

2.3.3 REGRAS UTILIZANDO XML

Em [VIEIRA], a linguagem utilizada para especificar um intruso foi o XML[10]. Na próxima seção haverá mais informações sobre essa poderosa linguagem. A estrutura de regras proposta por [VIEIRA] é a seguinte:

- *Alert:* Início de uma nova regra e possui o atributo *source* que representa a origem do dado. O atributo pode conter também os valores *log* e *message*;
- *Identification:* Representa um valor de identificação única da regra, análogo ao ID de um banco de dados;
- *RuleInformation:* Descreve a regra ou o evento intrusivo podendo informar os sistemas afetados;
- *Version:* Representa a versão da regra, auxiliando no controle das alterações;
- *Name:* Nome da regra;
- *Credits:* Autor ou a origem da regra;
- *Element:* Representa o elemento que pertence ao pacote de auditoria. Possui um atributo *value* que se refere ao nome do elemento e um valor para caracterizar a ocorrência do ataque. Uma regra pode ter mais de um *Element* para restringir a ocorrência;
- *Comprises:* Representa uma função lógica que proíbe um valor específico.
- *Range:* Representa uma função lógica que proíbe um intervalo de dados. Possui os atributos *MoreThan* e *LessThan* que define o intervalo superior e inferior, respectivamente.

A figura 2.3.3.1 mostra um exemplo de uma regra que detecta um ataque nos dados de auditoria provenientes da comunicação. Como regra necessária o

[10] XML: eXtensible Markup Language

elemento *service* precisa ter o valor WEB. O conteúdo proibido é um texto que contenha a string “XXX” em qualquer lugar da mensagem.

```
<alert source=”communication”>
        <identification>001</identification>
        <ruleInformation>
                This rule verify the inappropriate use
        </ruleInformation>
        <version>1.0.0</version>
        <name>inappropriateUse</name>
        <credits>LRG</credits>
        <element value="service">web</element>
        <comprises>XXX</comprises>
</alert>
```

FIGURA 2.3.3.1 REGRA QUE DETECTA UM ATAQUE ESCRITO NA LINGUAGEM XML.

2.4 LINGUAGEM XML

XML (extensible Markup Language) é uma linguagem de marcação recomendada pela *W3C* (World Wide Web Consortium) para gerar linguagens de marcação para necessidades especiais. É uma linguagem muito parecida com o *HTML* (Hyper Text Markup Language), a diferença é que o *XML* foi projetado para descrever os dados e dar ênfase em “o que” que aquele dado significa e o *HTML* é projetado para mostrar os dados e focar em como aquele dado deve aparecer.

O princípio do projeto era criar uma linguagem que pudesse ser ligada por software, e integrar-se com as demais linguagens. Sua filosofia seria incorporada por vários princípios importantes:

- Separação do conteúdo da formatação;
- Simplicidade de Legibilidade, tanto para humanos quanto para computadores;
- Criação de arquivos para validação da estrutura (DTD);

- Concentração na estrutura da informação e não na sua aparência.

Vale lembrar que *XML* é somente uma linguagem de marcação, o *XML* sozinho não realiza nenhuma ação. Por exemplo, observe o exemplo abaixo:

```
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

Existe as informações do remetente, destinatário, cabeçalho e corpo da mensagem. Mas se esse código *XML* não for inserido em um software, a mensagem não será enviada nunca.

Outra vantagem do *XML* é que os seus campos não são pré-definidos. Quem escreve o arquivo *XML* é quem deve definir os campos. Tornando viável a utilização dessa linguagem para definir as regras que especificam intrusos, pois caso haja necessidade de alteração só precisa alterar uma linha.

Como *XML* complementa os arquivos *HTML*, a integração de uma possível *wiki* contendo regras que especificam intrusão de vários administradores de ambientes *grid* será muito mais fácil. Além de que hoje em dia existem várias tecnologias que integram diferentes tipos de software utilizando *XML*.

2.4.1 DTD

O objetivo do *DTD* (Document Type Definition) é definir a construção dos blocos de um documento *XML*. Define a estrutura de um documento com uma lista de elementos “legais”. O *DTD* pode ser declarado dentro de um arquivo *XML* ou através de uma referência externa como segue o exemplo, mostrado na figura 2.4.1.1:

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note
(to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this
weekend!</body>
</note>
```

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

```
<?xml version="1.0"?>
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

FIGURA 2.4.1.1 A ESQUERDA DECLARAÇÃO DTD DENTRO DO ARQUIVO XML E O DO LADO DIREITO DEFINIDO EXTERNAMENTE.

3 PROPOSTA

Conforme visto nos capítulos anteriores, as linguagens que especificam detecção de intrusão não possuem uma boa documentação [DOYLE] ou não são viáveis em ambientes *grid* [SCHULTER] ou outros motivos. A proposta do trabalho é definir a descrição de ataques em ambientes *grid,* utilizando a linguagem *XML* (devido a sua alta capacidade de integração com outros softwares).

Com essa linguagem é possível realizar uma maior troca de informação, principalmente das regras, entre os Sistemas de Detecção de Intrusão de diferentes plataformas *grid*. Através dessas trocas os Sistemas de Detecção de Intrusão terão cada vez mais uma base de regras melhor elaborada, dificultando as ações dos intrusos.

Padronizando a linguagem de detecção de intrusão, as atualizações das regras dos IDS não precisam nem ser feitos através de um administrador. Podendo ser feito, por exemplo, através de algo similar ao *apt*[11].

Este capítulo está dividido da seguinte forma: a seção 3.1 aborda sobre a arquitetura; 3.2 define o que é um ataque; 3.3 define a estrutura de um arquivo XADG; 3.4 mostra alguns exemplos de um arquivo XADG; 3.5 aborda sobre como usufruir melhor do arquivo XADG em um ambiente *grid*.

3.1 ARQUITETURA

A arquitetura do XADG (eXtensible Attack Definition in Grid) está definida na figura 3.1:

[11] Um programa utilizado por algumas distribuições do linux que busca nos repositórios as atualizações disponíveis do sistema.

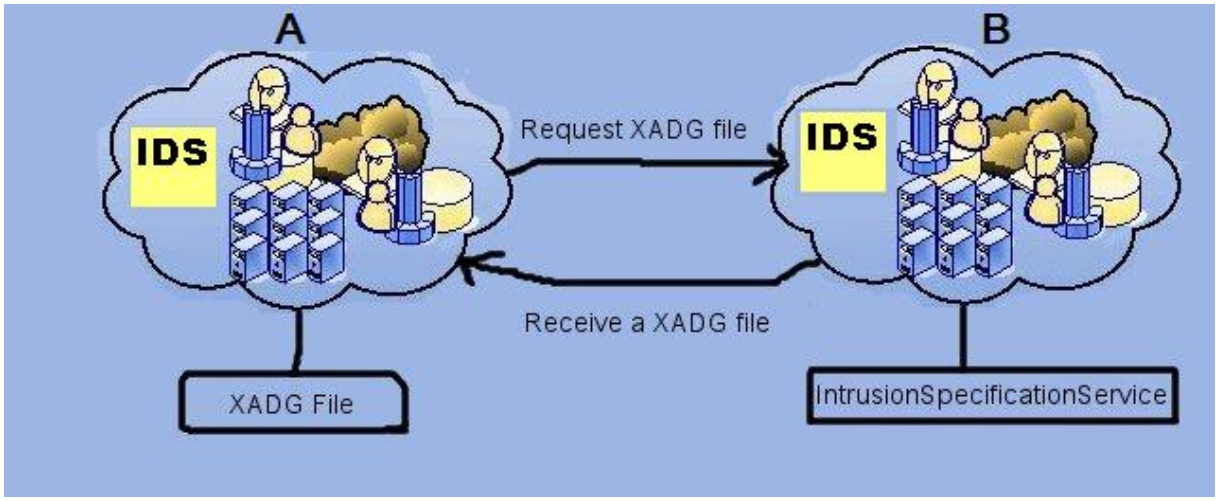


FIGURA 3.1 ARQUITETURA DO XADG DEMONSTRADO EM DOIS NODOS. NOTE QUE CADA NODO POSSUI O SEU PRÓPRIO IDS.

A figura 3.1 demonstra a arquitetura do XADG. Na figura demonstrada foi utilizado apenas dois nodos de um ambiente *grid* para facilitar o entendimento, mas poderia muito bem estar em um ambiente com dez nodos ou mais. Os passos a seguir descreve a arquitetura do XADG:

1. O administrador de um nodo, por exemplo A, possui um arquivo XADG muito antigo ou nem possui o mesmo e deseja obter esse arquivo;
2. O nodo pede para a *grid* o serviço de definição de ataque e recebe o nodo que possui o serviço desejado. O administrador faz uma requisição desse arquivo a esse outro nodo, por exemplo B;
3. Esse nodo retorna ao nodo solicitante o seu arquivo XADG;
4. Agora o nodo A analisa se deseja atualizar o arquivo (caso haja um desatualizado) e confirma a atualização.

3.2 DEFINIÇÃO DE ATAQUE

Por definição [VIEIRA] ataque é qualquer atividade maliciosa direcionada a um sistema ou serviço, podendo ou não torná-lo indisponível. Exemplos de ataques são: vírus, cavalos de tróia, vermes, *ping* da morte e entre outros. Porém, nem todos

os ataques são tão fáceis de serem identificado. Alguns ataques como *keyloger*, engenharia social ou má utilização de privilégios são descobertos somente depois que o dano já foi feito. Abaixo tem a definição de alguns ataques sobre acesso ilegítimo e negação de acesso segundo [KENDALL]:

- Engenharia social: Um *hacker* consegue acessar um sistema enganando um usuário autorizado, fazendo-o passar informações críticas para acessar um sistema ou entregar ao usuário algum cavalo de tróia para capturar informações como senha e endereços privados;
- Explorar falhas no sistema (*/exploits/*): Falhas de programação podem abrir espaço para que *hackers* explorem um acesso não autorizado do sistema. Problemas como */buffer/* e */overflow/* podem fazer o sistema executar um código malicioso;
- Abuso de privilégios: Um usuário que possui acesso ao sistema pode utilizar seus privilégios para executar ações impróprias como acessar jogos ou conteúdos proibido pelas políticas do sistema;
- Mau uso: Um usuário legítimo pode executar uma quantidade enorme de processos e consumir todo o recurso do sistema. Mesmo tendo acesso ao sistema essas práticas faz com que o consideremos um ataque, pois o sistema ficará indisponível.
- Obtenção de privilégios: Um usuário pode passar um código malicioso como parâmetro para o sistema e ao executar esse código o usuário passará a possuir mais privilégios do que ele deveria ter.
- DoS: Ataques de negação de serviço é quando um intruso tenta tornar um serviço indisponível e para isso utiliza diversas técnicas como sobrecarregar o sistema com solicitações de acesso. Um dos métodos para realização desse

ataque é: realizar requisições externas com tanta freqüência que force o computador alvo a reiniciar ou consumindo totalmente tornando-o indisponível.

3.3 ESTRUTURA DAS REGRAS DO XADG (eXtensible Attack Definition in Grid)

A proposta da estrutura do trabalho segue a estrutura das regras da linguagem proposta por [VIEIRA] com algumas modificações, pois já está na linguagem XML.

Os campos adicionado a estrutura de regras descrito na seção 2.3.3 são:

- *InvokeLimit*: um valor que limite o número de referências ao elemento durante um certo tempo;
- *Interval*: complementando a regra acima, o intervalo de tempo da limitação.
- *Time*: o horário em que a regra é válida, por padrão, seria o dia todo. Mas também existe a possibilidade de limitar um horário em que se aplica a regra.

3.4 EXEMPLOS DE REGRAS

Nas figura 3.4.1 segue um exemplo de regras utilizando a linguagem XADG:

```
<alert source=”communication”>
        <identification>rule1</identification>
        <ruleInformation>
                This rule verify the inappropriate use
        </ruleInformation>
        <version>1.0.0</version>
        <name>inappropriateUse</name>
        <credits>LRG</credits>
```

```
        <element value="service">web</element>
        <comprises>porn</comprises>
</alert>
```

FIGURA 3.4.1 EXEMPLO DE REGRA QUE DEFINE INTRUSO.

A regra acima é uma regra que atende o ataque descrito na seção anterior como **mau uso**. Uma regra que é necessária é que o elemento *service* possua o valor *web*, a regra não permite que ocorra a transmissão de qualquer texto contendo a palavra “porn”.

A figura 3.4.2 segue outro exemplo de regras para evitar intrusão no sistema:

```
<alert source=”communication”>
        <identification>rule2</identification>
        <ruleInformation>
                This rule verify the DoS
        </ruleInformation>
        <version>1.0.0</version>
        <name>dos</name>
        <credits>LRG</credits>
        <element value="service">web</element>
        <invokelimit>10</invokelimit >
        <interval>60</interval>
</alert>
```

FIGURA 3.4.2 EXEMPLO DE REGRAS PARA VERIFICAR ATAQUE DE DOS.

A regra acima tenta evitar o ataque conhecido como **DoS**. A regra diz que o elemento *service* que possui o valor *web* não pode ser referenciado mais de 10 vezes, num limite de 60 segundos.

A figura 3.4.3, segue mais um exemplo:

```
<alert source=”communication”>
        <identification>rule3</identification>
        <ruleInformation>
                This rule verify the bad use
        </ruleInformation>
        <version>1.0.0</version>
        <name>dos</name>
        <credits>LRG</credits>
        <element value="service">storeSqlDb</element>
        <time>
                <start>08:00:00</start>
                <end>12:00:00</end>
        </time >
</alert>
```

FIGURA 3.4.3 EXEMPLO DE REGRAS PARA VERIFICAR ATAQUE DE MAU USO.

A regra acima estipula que o serviço storeSqlDb (possivelmente um serviço de armazenamento de banco de dados) esteja disponível para o grid das 08:00:00 ao 12:00:00. Pois em outros horários esse nodo é utilizado para outras tarefas.

3.5 REGRAS ESPECÍFICAS PARA CADA AMBIENTE

Para o bom funcionamento de um IDS baseado em regras, é necessário que as regras sejam elaboradas da forma mais específica possível. Não teria sentido possuir regras tão rígidas num ambiente *grid* em casa ou ter um ambiente de trabalho sem muitas regras.

Em um ambiente doméstico a intenção principal do usuário poderia ser, por exemplo, possuir a maior liberdade possível. Sem precisar se incomodar em ter que ficar digitando senhas pra tudo que ele queira fazer. Enquanto que num ambiente de trabalho, também não pode ser tão liberal.

Dependendo do ambiente que os usuários desejarem, um sistema de regra deve ser elaborado para que o mesmo seja o mais seguro e mais otimizado possível.

4. IMPLEMENTAÇÃO

Com o intuito de testar as funcionalidades do *IntrusionSpecificationService*, este capítulo apresenta o resultado da implementação do serviço. O serviço consiste em implementar a troca do arquivo XADG entre dois nodos, verificar as diferenças entre os dois arquivos (quando existir) e pedir a confirmação do administrador.

Esse capítulo se subdivide da seguinte forma: a seção 4.1 mostra o diagrama de seqüência do serviço; 4.2 mostra as principais partes do resultado do serviço; 4.3 aborda sobre a validação do trabalho.

4.1 DIAGRAMA DE SEQÜÊNCIA

Abaixo será descrito os passos que ocorre quando o administrador do nodo solicita a atualização do seu arquivo XADG:

1. O administrador do nodo requerente pede ao nodo(s) adjacente que necessita do serviço (*IntrusionSpecificationService*);
2. Os nodos se interagem entre si até encontrar um nodo que possui o serviço requisitado;
3. Os dois nodos (nodo requerente e o nodo fornecedor) estabelecem uma conexão direta;
4. O nodo requerente envia uma *Task* (passando o seu arquivo XADG e o serviço *updateXadgFile*) para o nodo fornecedor que retorna um TaskResult (retornando o arquivo XADG);
5. No nodo é feita a verificação entre os dois arquivos XADG ;
6. Caso o administrador ache viável a atualização ele confirma a operação e o seu arquivo XADG será atualizado.

Para tentar demonstrar melhor a implementação, foi feito um diagrama de seqüência. A figura 4.1 mostra o diagrama de seqüência:

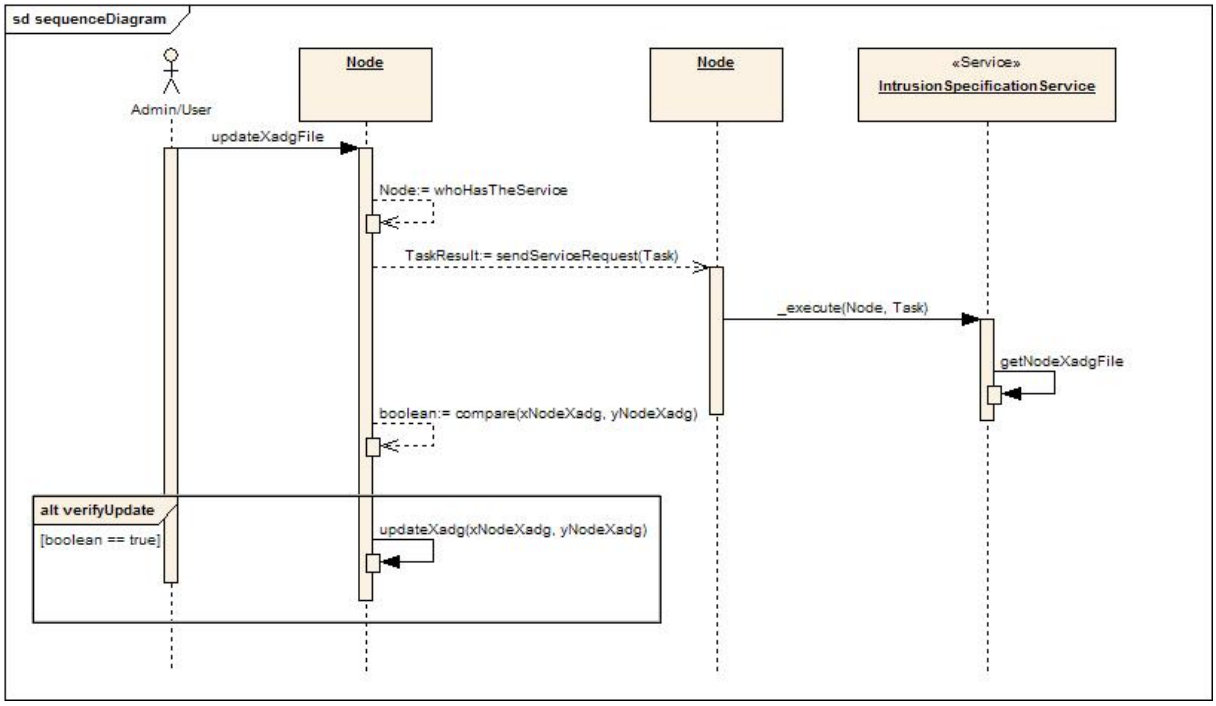


FIGURA 4.1 DIAGRAMA DE SEQÜÊNCIA DO INTRUSIONSPECIFICATIONSERVICE.

4.2 GRID-M

O *grid-m* é uma plataforma utilizada para construir aplicações de computação em *grid* e aplicações em dispositivos embargados e móveis. Essa plataforma fornece uma Application Programming Interface (API) para conectar aplicações desenvolvidas em Java em um ambiente de computação em *grid*. Seu perfil *runtime* é pequeno bastante para ser usado em aplicações de computação móvel. Acredita-se que as soluções do *grid-m* respondam à pergunta sobre uma plataforma para promover a homogeneização neste ambiente de desenvolvimento [FRANKE].

4.3 RESULTADOS

A plataforma escolhida para a validação do trabalho foi o *grid-m*, a principal justificativa para essa escolha é porque o produto foi desenvolvido por integrantes

do laboratório LRG. Facilitando assim, o acesso ao código fonte, o contato com os desenvolvedores e utilização do XADG na plataforma.

O código completo da classe principal e a saída da mesma está em anexo (anexo I, anexo II e anexo III). Mas as partes mais relevantes da saída será mostrado resumidamente na figura 4.3.1 e figura 4.3.2:

```
<task-result>
<result-code>OK</result-code>
<originator>node-2</originator>
<destination>node-0</destination>
<taskid>task-1000</taskid>
<timestamp>1190410773625</timestamp>
<parameters>
<alert>
<identification>rule1</identification>
<ruleInformation>
This rule verify the inappropriate use
</ruleInformation>
<version>1.0.0</version>
<name>inappropriateUse</name>
<credits>LRG</credits>
<element>web</element>
<comprises>web</comprises>
</alert>
</parameters>
</task-result>
```

FIGURA 4.3.1 SAÍDA DA TASKRESULT.

A figura 4.3.1 mostra a *TaskResult* que o nodo requerente recebe o arquivo XADG do nodo fornecedor.

>>>> INSERT BEFORE 1

<parameters>

>>>> On line 3, CHANGED FROM

<ruleInformation>

This rule verify the inappropriate use

</ruleInformation>

>>>> CHANGED TO

<ruleInformation>This rule verify the inappropriate use</ruleInformation>

>>>> On line 9, CHANGED FROM

<element value="service">web</element>

>>>> CHANGED TO

<element>web</element>

>>>> INSERT BEFORE 12

</parameters>

>>>> End of differences.

>>>> INSERT BEFORE 1

<parameters>

>>>> On line 3, CHANGED FROM

<ruleInformation>

>>>> On line 4, CHANGED FROM

This rule verify the inappropriate use

>>>> On line 5, CHANGED FROM

</ruleInformation>

>>>> CHANGED TO

<ruleInformation>This rule verify the inappropriate use</ruleInformation>

>>>> On line 9, CHANGED FROM

<element value="service">web</element>

>>>> CHANGED TO

<element>web</element>

>>>> INSERT BEFORE 12

</parameters>

Would you want to update your XADG file? (Y/N)

Y

The node's XADG file is updated.

FIGURA 4.3.2 DIFERENÇA ENTRE OS ARQUIVOS XADG DOS DOIS NODOS.

A figura 4.3.2 mostra a diferença entre os arquivos XADG dos dois nodos.

4.4 VALIDAÇÃO

A principal contribuição desse trabalho é a proposta do arquivo XADG que proporcionará uma segurança adicional ao ambiente *grid*. A implementação mostrada prova que com a padronização do arquivo que define um intruso em um ambiente *grid*, auxiliará bastante a vida de um administrador de um nodo, fazendo com que ele possa se concentrar em outras questões de seguranças que ainda não são solucionáveis pelas máquinas (ou são solucionáveis de uma forma ineficiente).

5 CONCLUSÃO

Como em todo sistema de segurança é difícil prever a “criatividade” dos maliciosos e conseguir fazer um sistema de regras que além de prover uma boa segurança, ainda consiga tornar o ambiente agradável. Um bom exemplo é o sistema de alarme domiciliar.

Em relação a segurança, conseguimos agir de uma maneira mais eficiente se conhecemos um caso similar que já tenha acontecido. Podendo assim, tomar as medidas preventivas para que o mesmo caso não se repita.

Como a computação em *grid* ainda está em evolução. Ainda não existem tantos relatos em relação a mais novos tipos de ataques e também porque esse tipo de computação, no momento, só é utilizado por pessoas que são da área da computação.

Justamente essa falta de registros de ataques na computação em *grid* foi a maior dificuldade encontrada durante o trabalho para elaboração de regras que consigam prevenir os ataques mais sofisticados.

Mas com a realização dos trabalhos futuros, acredito que este trabalho consiga contribuir de uma forma positiva para a computação em *grid*. Ajudando na sua evolução em relação a segurança e que futuramente mais usuários a utilizem.

6 TRABALHOS FUTUROS

Os trabalhos futuros que podem ser citados são:

- Para facilitar a utilização do XADG, seria interessante manter paralelamente um *site*, análogo aos sistemas de *wiki* ou *blog* para facilitar a comunicação entre elaboradores de regras XADG e integrá-lo ao *grid-m*. Pois havendo um compartilhamento entre mais elaboradores de regras, todos saem ganhando. O trabalho de conclusão de [CONTE 2007] aborda esse assunto;
- Deixar rodando o XADG em um ambiente *grid* que utiliza plataformas *grid* diferentes, juntamente com o IDS e registrar os ataques para que futuramente possa ser elaborada regras que consigam preveni-las e que tente manter a mesma eficiência;
- Implementar um serviço que elabore regras utilizando XADG baseado em logs de um IDS (utilizando alguma heurística).
- Testar compatibilidade do serviço com outras plataformas.

REFERÊNCIAS BIBLIOGRÁFICAS

ALLEN, Julia et al. *State of the Practice of Intrusion Detection Technologies*. Technical Report CMU/SEI-99-TR-028, Software Engineering Institute, Carnie Mellon University, jan. 2000.

Association of Lisp Users. Disponível em: http://lisp.org/alu/home. Acesso em: 20 nov. 2007.

AXELSSON, Stefan. *Research in Intrusion-Detection Systems: A Survey*. Technical Report TR-98-17, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, aug. 1999. 93 p.

BARBOSA, André. *Sistemas de Detecção de Intrusão*. Rio de Janeiro: COPPE/UFRJ, 2000.

BUYYA, Rajkumar. *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. 2002. Melbourne, Australia: Monash University, 2002.

Chapter 2 Writing snort rules. How to write snort rules and keep your sanity. Disponível em: http://www.snort.org/docs/writing_rules/chap2.html. Acesso em: 5 out. 2007.

CHRYSSOVERGIS, Marcelo Digiacomo. *Um componente para verificação de políticas de segurança no Globus Toolkit 3*. Florianópolis: UFSC, 2006.

CONTE, Ezequiel. *Intranet na CERTI: Um modelo baseado em recursos de Enterprise 2.0*. Florianópolis: UFSC, 2007.

CORNELLI, F., et al. *Choosing Reputable Servents in a P2P Network*, In: Eleventh International World Wide Web Conference, Honolulu, Hawaii, pp. 376-386, 2002.

DANTAS, Mario. *Computação Distribuída de alto desempenho: Redes, clusters e grids computacionais.* Axcel Books, 2005.

DOYLE, Jon. *Some Representational Limitations of the Common Intrusion Specification Language*. Cambridge: MIT, 1999.

FEIERTAG, R. A Common Intrusion Specification Language (CISL). Disponível em: http://gost.isi.edu/cidf/drafts/language.txt. Acesso em: 10 abr. 2007.

FOSTER, I. *Internet Computing and the Emerging Grid*. *Nature Web Matters*, 2000. Disponível em: http://www.nature.com/nature/webmatters/grid/grid.html. Acesso em: 11 fev. 2007.

FRANKE, Hans. *Grid-m: Middleware para integrar dispositivos móveis, sensores e computação em grid*. Florianópolis: UFSC, 2007.

KENDALL, Kristopher. *A Database of Computer Attacks for the Evalution Intrusion Detection Systems*. Cambridge: MIT, 1999.

SCHULTER, Alexandre. *Integração de Sistemas de Detecção de Intrusão para Segurança de Grids Computacionais*. Florianópolis: UFSC, 2006.

TANENBAUM, Andrew S. *Distributed System: Principles and Paradigms*. 1. st. edition. **Prentice Hall**, 2002.

VIEIRA, Kleber. *Mecanismos para aplicação paralela de técnicas distintas de detecção de intrusão em ambientes de grid*. Florianópolis: UFSC, 2007.

Xml Tutorial. Disponível em: http://www.w3schools.com/xml/default.asp. Acesso em: 05 out. 2007.

ANEXO I – CÓDIGO FONTE (Main.Java)

```
package XADG.test;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import gridm.HTTPServerInterface;
import gridm.NetInterface;
import gridm.Node;
import gridm.Task;
import gridm.TaskResult;
import gridm.XMLTree;

import XADG.IntrusionSpecificationService;
import XADG.StringToXml;
import XADG.XmlToXMLTree;

/**
 * TEST SCENARIO:
 *
 *   Node-0 sends sends without knowing where is the service
 *
 * +------+    +------+    +------+
 * |node-0|<==>|node-1|<==>|node-2|-- IntrusionSpecificationService
 * +------+    +------+    +------+
 *
 *
 * - three nodes: node-0, node-1, node-2 each one just knowing about neighbor
```

```
 * - node-2 has MultiplyService attached to it.
 * - node-0 submits task to *
 *
 * RESULTS:
 * - node-0 ask to node-1 about service
 * - node-1 ask to node-2 about service *
 * - node-2 reply to node-1: "i have this service"
 * - node-1 reply to node-0: "node-2 has the service"
 * - node-0 added direct route to node-2
 * - node-0 sends task to node-2
 * - node-2 sends task result to node-0
 *
 * @author Hung Ruo Han
 *
 */

public class Main {
	public static void main(String[] args) {
		// Set node-0
		Node node0 = new Node("node-0", "http://localhost:8000");
		node0.setLogLevel(5);
		NetInterface httpNetInterface = new HTTPServerInterface(node0, 8000);
		httpNetInterface.setSnoopMode(true);
		node0.setNetInterface(httpNetInterface);
		node0.start();

		// Set node-1
		Node node1 = new Node("node-1", "http://localhost:8001");
		node1.setLogLevel(5);
		httpNetInterface = new HTTPServerInterface(node1, 8001);
```

```
httpNetInterface.setSnoopMode(true);
node1.setNetInterface(httpNetInterface);
node1.start();

// Set node-2
Node node2 = new Node("node-2", "http://localhost:8002");
node2.setLogLevel(5);
node2.addService("updateXadgFile", new IntrusionSpecificationService());
httpNetInterface = new HTTPServerInterface(node2, 8002);
httpNetInterface.setSnoopMode(true);
node2.setNetInterface(httpNetInterface);
node2.start();

// Create route table
node0.addRoute("node-1", "http://localhost:8001");
node1.addRoute("node-0", "http://localhost:8000");
node1.addRoute("node-2", "http://localhost:8002");
node2.addRoute("node-1", "http://localhost:8001");

// Create a new task invoking updateXadgFile service
// Ooppss.. we dont know where this service runs. Dont care! ;)
Task task = new Task("*", "updateXadgFile", null);

// Set parameters
String myNodePath = "D:\\UFSC\\TCC\\ids\\Implementacao\\node0.xadg";

XmlToXMLTree myTree = new XmlToXMLTree (myNodePath);
XMLTree xTree = myTree.getXmlTree();

task.addParameters(xTree);
```

```
// Send task
TaskResult result = node0.sendServiceRequest(task);

// Get the XMLTree (result)
XMLTree resultTree = result.getParameters();
String otherNodeXml = resultTree.toString();
System.out.println("Return: Received the other node's XADG file.");

// Save the other node's XADG file in a .xadg to compare with my own XADG file
String otherNodePath = "D:\\UFSC\\TCC\\ids\\Implementacao\\otherNode.xadg";
StringToXml stxOther = new StringToXml (otherNodeXml, otherNodePath);
stxOther.stringToFile();

// Receive a String which have the difference between the two XADG files
String diffText = analyseDifference (myNodePath, otherNodePath);
System.out.println (diffText);

// Ask if the use/admin want to update your XADG file
System.out.println ("Would you want to update your XADG file? (Y/N)");

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
String input = "";

try {
        input = in.readLine();
} catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

```
                input = input.toUpperCase();

                // If the user/admin decide to update, then save the new XADG file
                if ( input.equals ("Y")) {
                        StringToXml stx = new StringToXml (otherNodeXml, myNodePath);
                        stx.stringToFile();
                        System.out.println ("The node's XADG file is updated.");
                }
        }

        /**
         * A method which analyse the difference between two files (references)
         *
         * @param myNodePath
         * @param otherNode
         * @return differenceString
         */
        private static String analyseDifference(String myNodePath, String otherNode) {
                // TODO Auto-generated method stub
                Diff diff = new Diff();
                diff.doDiff(myNodePath, otherNode);

                return diff.getDifference();
        }
}
```

ANEXO II – CÓDIGO FONTE (IntrusionSpecificationService.Java)

```
package XADG;


import gridm.Node;
import gridm.Service;
import gridm.Task;
import gridm.TaskResult;
import gridm.XMLTree;


/**
 * SIMPLE IntrusionSpecificationService that process 'updateXadgFile' service request.
 *
 * CREATING A TASK FOR THIS SERVICE:
 *    Task task = new Task("node-X", "updateXadgFile", null);
 *    task.addParameters(XMLTree myXadgFile);
 *
 * @author Hung Ruo Han
 *
 */
public class IntrusionSpecificationService extends Service {
        private String myNodePath;


        /**
         * Real running code for this service.
         *
         * @param node that trigged this hook
         * @param task to be serviced
         * @return processing result
         * @throws java.lang.Exception for any execution exception
```

```
     * @author Hung Ruo Han
     *
     */
    protected TaskResult _execute(Node node, Task task) throws Exception {
        // COLLECT PARAMETERS
        XMLTree params = task.getParameters();
        String textXml = params.toString();

        // PASSING A XMLTREE TO A XADG FILE
        String otherNodePath = "D:\\UFSC\\TCC\\ids\\Implementacao\\analizeNode.xadg";
        new StringToXml (textXml, otherNodePath);

        // SET THE NODE'S XADG FILE PATH
        myNodePath = "D:\\UFSC\\TCC\\ids\\Implementacao\\node2.xadg";

        // PREPARING TO SAVE THE RESULT...
        TaskResult result = null;

        // PASS THE NODE'S XADG FILE TO THE REQUESTER NODE
        if ( task.getService().equals("updateXadgFile") ) {
            XmlToXMLTree myNodeXml = new XmlToXMLTree (myNodePath);
            XMLTree myTree = myNodeXml.getXmlTree();
            result = new TaskResult(task, TaskResult.RESULT_OK, null);
            result.addParameters(myTree);
        }

        // RETURN THE NODE'S XADG FILE ON A XMLTREE
        return result;
    }
}
```

ANEXO III – Saída

[00000140] 5 node-0 Note HTTPServer http-server addServlet(1) address=/

[00000140] 5 node-0 Note HTTPServer http-server addServlet(1) address=/request

[00000140] 5 node-0 Note HTTPServer http-server addServlet(1) address=/response

[00000000] 5 node-1 Note HTTPServer http-server addServlet(1) address=/

[00000000] 5 node-1 Note HTTPServer http-server addServlet(1) address=/request

[00000000] 5 node-1 Note HTTPServer http-server addServlet(1) address=/response

[00000000] 5 node-2 Note HTTPServer http-server addServlet(1) address=/

[00000016] 5 node-2 Note HTTPServer http-server addServlet(1) address=/request

[00000032] 5 node-2 Note HTTPServer http-server addServlet(1) address=/response

[00000172] 5 node-0 Note RouteAdd networker addRoute(1) host=node-1:url=http://localhost:8001:metric=1

[00000032] 5 node-1 Note RouteAdd networker addRoute(1) host=node-0:url=http://localhost:8000:metric=1

[00000032] 5 node-1 Note RouteAdd networker addRoute(1) host=node-2:url=http://localhost:8002:metric=1

[00000032] 5 node-2 Note RouteAdd networker addRoute(1) host=node-1:url=http://localhost:8001:metric=1

[00000172] 5 node-0 Note DiscoveryTask node sendServiceRequest(1) trying to discovery service updateXadgFile requested by node-0

[00000172] 5 node-0 Note RouteTableRequested networker getRouteTable(1) returning route table of node-0

[00000172] 5 node-0 Note RouteFound networker findRoute(1) host=node-1:url=http://localhost:8001:found in route table

[00000250] 5 node-0 Note HTTPSending http-server httpClient(1) url=http://localhost:8001/request:size=334

<== HTTP CLIENT node-0 SENDING REQUEST (to http://localhost:8001/request)

POST /request http

<task>

<service>discovery</service>

<originator>node-0</originator>

<destination>node-1</destination>

<taskid>task-1001</taskid>

<timestamp>1190410772125</timestamp>

<priority>2</priority>

<parameters>

<discovery-parameters>

<service>updateXadgFile</service>

<requester>node-0</requester>

</discovery-parameters>

</parameters>

</task>

[00000922] 5 node-1 Note http-servlet HTTPServletReceived doPost(2) task=Task(task-1001,discovery,node-0,node-1,...)

[00000938] 5 node-1 Note ReceivedDiscoveryTask node-1 processServiceRequest(0) node-1 service updateXadgFile requested by node-0 not found! forwarding discovery task...

[00000938] 5 node-1 Note StoreInTaskStak networker StoreTaskStack(1) task=task-1001:queueSize=1

[00000938] 5 node-1 Note DiscoveryTask node sendServiceRequest(1) trying to discovery service updateXadgFile requested by node-0

[00000938] 5 node-1 Note RouteTableRequested networker getRouteTable(1) returning route table of node-1

[00000938] 5 node-1 Note RouteFound networker findRoute(1) host=node-2:url=http://localhost:8002:found in route table

[00000938] 5 node-1 Note HTTPSending http-server httpClient(1) url=http://localhost:8002/request:size=334

<== HTTP CLIENT node-1 SENDING REQUEST (to http://localhost:8002/request)

POST /request http

<task>

<service>discovery</service>

<originator>node-1</originator>

<destination>node-2</destination>

<taskid>task-1002</taskid>

<timestamp>1190410773031</timestamp>

<priority>2</priority>

<parameters>

<discovery-parameters>

<service>updateXadgFile</service>

<requester>node-0</requester>

</discovery-parameters>

</parameters>

</task>

[00000953] 5 node-2 Note http-servlet HTTPServletReceived doPost(2) task=Task(task-1002,discovery,node-1,node-2,...)

[00000953] 5 node-2 Note ReceivedDiscoveryTask node-2 processServiceRequest(0) found service updateXadgFile requested by node-0 in this node

[00000953] 5 node-2 Note http-servlet HTTPServletResponding doPost(4) task=Task(task-1002,discovery,node-1,node-2,...):taskresult=TaskResult(task-1002,OK,node-2,node-1,...):status=200

[00001438] 5 node-1 Note HTTPReceiving http-server httpClient(2) result=200:message=OK:size=372

[00001453] 5 node-1 Note HTTPSending http-server sendTask(4) done:taskResult for task-1002 stored: taskResult.resultCode=OK: queuesize=1

[00001453] 5 node-1 Note RemoveFromTaskStack networker RemoveTaskStack(1) task=task-1001:queueSize=0

[00001453] 5 node-1 Note http-servlet HTTPServletResponding doPost(4) task=Task(task-1001,discovery,node-1,*,...):taskresult=TaskResult(task-1001,OK,node-1,node-0,...):status=200

==> HTTP CLIENT node-1 RECEIVED RESPONSE (for http://localhost:8002/request)

200 OK

<task-result>

<result-code>OK</result-code>

<originator>node-2</originator>

<destination>node-1</destination>

<taskid>task-1002</taskid>

<timestamp>1190410773046</timestamp>

<parameters>

<result-discovery>

<service>updateXadgFile</service>

<node>node-2</node>

<url>http://localhost:8002</url>

<requester>node-0</requester>

</result-discovery>

</parameters>

</task-result>

==> HTTP CLIENT node-0 RECEIVED RESPONSE (for http://localhost:8001/request)

200 OK

<task-result>

<result-code>OK</result-code>

<originator>node-1</originator>

<destination>node-0</destination>

<taskid>task-1001</taskid>

<timestamp>1190410773046</timestamp>

<parameters>

<result-discovery>

<service>updateXadgFile</service>

<node>node-2</node>

<url>http://localhost:8002</url>

<requester>node-0</requester>

</result-discovery>

</parameters>

</task-result>

<== HTTP CLIENT node-0 SENDING REQUEST (to http://localhost:8002/request)

POST /request http

<task>

<service>ping</service>

<originator>node-0</originator>

<destination>node-2</destination>

<taskid>task-1003</taskid>

<timestamp>1190410773546</timestamp>

<priority>2</priority>

</task>

==> HTTP CLIENT node-0 RECEIVED RESPONSE (for http://localhost:8002/request)

200 OK

<task-result>

<result-code>OK</result-code>

<originator>node-2</originator>

<destination>node-0</destination>

<taskid>task-1003</taskid>

<timestamp>1190410773578</timestamp>

<parameters>

<result>pong</result>

</parameters>

</task-result>

[00001593] 5 node-0 Note HTTPReceiving http-server httpClient(2) result=200:message=OK:size=372

[00001593] 5 node-0 Note HTTPSending http-server sendTask(4) done:taskResult for task-1001

stored: taskResult.resultCode=OK: queuesize=1

[00001593] 5 node-0 Note DiscoveryTask node sendServiceRequest(2) discovered how to find service

updateXadgFile => node-2

[00001593] 5 node-0 Note PingNode networker ping(1) sending ping to node-2

[00001593] 5 node-0 Note HTTPSending http-server httpClient(1)

url=http://localhost:8002/request:size=191

<== HTTP CLIENT node-0 SENDING REQUEST (to http://localhost:8002/request)

POST /request http

<task>

<service>updateXadgFile</service>

<originator>node-0</originator>

<destination>node-2</destination>

<taskid>task-1000</taskid>

<timestamp>1190410772125</timestamp>

<priority>2</priority>

<parameters>

<alert>

<identification>rule1</identification>

<ruleInformation>

This rule verify the inappropriate use

</ruleInformation>

<version>1.0.0</version>

<name>inappropriateUse</name>

<credits>LRG</credits>

<element>web</element>

<comprises>web</comprises>

</alert>

</parameters>

</task>

[00001485] 5 node-2 Note http-servlet HTTPServletReceived doPost(2) task=Task(task-1003,ping,node-0,node-2,...)

[00001485] 5 node-2 Note pingReceived node-2 processServiceRequest(0) pong reply to node-0

[00001485] 5 node-2 Note http-servlet HTTPServletResponding doPost(4) task=Task(task-1003,ping,node-0,node-2,...):taskresult=TaskResult(task-1003,OK,node-2,node-0,...):status=200

[00001625] 5 node-0 Note HTTPReceiving http-server httpClient(2) result=200:message=OK:size=238

[00001625] 5 node-0 Note HTTPSending http-server sendTask(4) done:taskResult for task-1003 stored: taskResult.resultCode=OK: queuesize=2

[00001625] 5 node-0 Note RouteAdd networker addRoute(1) host=node-2:url=http://localhost:8002:metric=1

[00001625] 5 node-0 Note RouteFound networker findRoute(1) host=node-2:url=http://localhost:8002:found in route table

[00001625] 5 node-0 Note HTTPSending http-server httpClient(1) url=http://localhost:8002/request:size=490

[00001532] 5 node-2 Note http-servlet HTTPServletReceived doPost(2) task=Task(task-1000,updateXadgFile,node-0,node-2,...)

java.io.FileNotFoundException: D:\UFSC\TCC\ids\Implementacao\node2.xadg (The system cannot find the file specified)

at java.io.FileInputStream.open(Native Method)

at java.io.FileInputStream.<init>(Unknown Source)

```
        at java.io.FileInputStream.<init>(Unknown Source)
        at java.io.FileReader.<init>(Unknown Source)
        at XADG.XmlToString.readFileAsString(XmlToString.java:38)
        at XADG.XmlToString.<init>(XmlToString.java:23)
        at XADG.XmlToXMLTree.getXmlTree(XmlToXMLTree.java:27)
        at XADG.IntrusionSpecificationService._execute(IntrusionSpecificationService.java:50)
        at gridm.Service.processService(Service.java:46)
        at gridm.Node.processServiceRequest(Node.java:968)
        at gridm.Networker.receiveTask(Networker.java:464)
        at gridm.HTTPServerInterfaceServlet.doPost(HTTPServerInterfaceServlet.java:114)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:760)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
        at minihttp.HTTPServer.forward(HTTPServer.java:404)
        at minihttp.HTTPConnectionHandler.run(HTTPConnectionHandler.java:78)
        at java.lang.Thread.run(Unknown Source)
[00001532] 5 node-2 Note http-servlet HTTPServletResponding doPost(4) task=Task(task-
1000,updateXadgFile,node-0,node-2,...):taskresult=TaskResult(task-1000,OK,node-2,node-
0,...):status=200
[00001672] 5 node-0 Note HTTPReceiving http-server httpClient(2) result=200:message=OK:size=478

==> HTTP CLIENT node-0 RECEIVED RESPONSE (for http://localhost:8002/request)
200 OK

<task-result>
<result-code>OK</result-code>
<originator>node-2</originator>
<destination>node-0</destination>
<taskid>task-1000</taskid>
<timestamp>1190410773625</timestamp>
<parameters>
```

<alert>

<identification>rule1</identification>

<ruleInformation>

This rule verify the inappropriate use

</ruleInformation>

<version>1.0.0</version>

<name>inappropriateUse</name>

<credits>LRG</credits>

<element>web</element>

<comprises>web</comprises>

</alert>

</parameters>

</task-result>

Return: Received the other node's XADG file.

[00001687] 5 node-0 Note HTTPSending http-server sendTask(4) done:taskResult for task-1000

stored: taskResult.resultCode=OK: queuesize=3

>>>> Difference of file "D:\UFSC\TCC\ids\Implementacao\node0.xadg" and file

"D:\UFSC\TCC\ids\Implementacao\otherNode.xadg".

>>>> INSERT BEFORE 1

<parameters>

>>>> On line 3, CHANGED FROM

<ruleInformation>

This rule verify the inappropriate use

</ruleInformation>

>>>> CHANGED TO

<ruleInformation>This rule verify the inappropriate use</ruleInformation>

>>>> On line 9, CHANGED FROM

<element value="service">web</element>

>>>> CHANGED TO

<element>web</element>

>>>> INSERT BEFORE 12

</parameters>

>>>> End of differences.

>>>> INSERT BEFORE 1

<parameters>

>>>> On line 3, CHANGED FROM

<ruleInformation>

>>>> On line 4, CHANGED FROM

This rule verify the inappropriate use

>>>> On line 5, CHANGED FROM

</ruleInformation>

>>>> CHANGED TO

<ruleInformation>This rule verify the inappropriate use</ruleInformation>

>>>> On line 9, CHANGED FROM

<element value="service">web</element>

>>>> CHANGED TO

<element>web</element>

>>>> INSERT BEFORE 12

</parameters>

Would you want to update your XADG file? (Y/N)

Y

The node's XADG file is updated.

ANEXO IV – Artigo

Proposta de descrição de ataques em ambientes grid

Hung R. Han[1], Kleber M. M. Vieira[1], Fernando Koch[1], Carla M. Westphall[1], Carlos B. Westphall[1]

Laboratório de Redes e Gerência - LRG

[1]Departamento de Informática e Estatística - INE

Universidade Federal de Santa Catarina (UFSC)

Florianópolis – Santa Catarina – Brasil

{hung,kleber,koch,carlamw,westphal}@inf.ufsc.br

Abstract. *The specialized literature suggests that techniques for grid computing intrusion detection are a solution to improve the security in these environments. However, the lack of a standard language to describe intrusion is a shortcoming in current works. This work aims to provide a standard definition. We argue this standardization will ease the administration efforts, promote knowledge sharing and, ultimately, help with the technology's dissemination*

Resumo. *Com o crescente esforço para incrementar a segurança em sistemas de computação em grid, técnicas de detecção de intrusão vem sendo sugerida pela literatura. Este trabalho tem por objetivo padronizar a descrição de ataques de intrusões nesse tipo de ambiente, pois até agora as linguagens conhecidas que abordam o assunto possuem uma documentação ruim (devido ao abandono do projeto) ou não são viáveis para um ambiente de computação em grid. A proposta deste trabalho é padronizar uma descrição que defina um ataque num ambientes de computação em grid. Com essa padronização, será mais fácil para os administradores de ambientes em* grid *compartilharem informações, tornando esse tipo de ambiente cada vez mais seguro, e por conseqüência, tendo mais usuários querendo utilizar essa tecnologia.*

Palavras-chave*: Computação em grid. Sistema de Detecção de Intrusão. Especificação de intrusão.*

1. Introdução

Hoje em dia a internet se tornou um meio de comunicação amplamente utilizado. O próximo passo na evolução da internet seria integrar vários computadores entre si

com o objetivo de aproveitar ao máximo o desempenho e serviços disponíveis por eles.

A proposta do trabalho de conclusão de curso é propor as regras que definem um intruso num ambiente grid, através de uma linguagem escolhida. Contribuindo assim, para a melhora do Sistema de Detecção de Intrusão (IDS – *Intrusion Detection System*). A idéia é que essas regras sejam bastante utilizadas em diversas plataformas *grid*, tornando assim a linguagem e as regras padrão para especificação de intrusos nesse tipo de ambiente.

Com essa possível padronização administradores e consultores não precisariam estar aprendendo uma linguagem nova de especificação de regras de intrusão cada vez que eles entrarem em contato com uma nova plataforma *grid*. Além disso, é possível que administradores de diferentes plataformas *grid* consigam trocar experiências, fazendo com que todos consigam usar uma nomenclatura em comum. Assim os sistemas de ambiente em *grid* tornam-se cada vez mais seguros, beneficiando os usuários finais.

Como as questões de segurança na área da computação são muito complexo (devido a má intenção, falta de informação, inexperiência de alguns usuários e outros motivos) fica difícil de conseguir fazer com que os usuários se sintam seguros ao utilizar os serviços computacionais, por exemplo, hoje em dia muitos usuários ainda não se sentem seguros em pagar as suas contas via internet.

A partir dessa motivação, o trabalho tem por objetivo dar uma contribuição para a melhoria da segurança da computação *grid*. Por conseqüência, contribuindo para a evolução da computação nesse tipo de ambiente. Pois quanto mais um usuário se sente seguro ao utilizar o sistema, mais ele vai utilizar esse sistema. O usuário gostando de utilizar o sistema é possível que ele divulgará para os seus amigos e colegas de

trabalhos, esses por sua vez, falaram para os próximos amigos. Conseqüentemente, aumentará a quantidade de usuários que utilizam esse sistema.

Este artigo está organizado em 6 seções: seção 2 aborda os fundamentos teóricos necessários para tratar do assunto deste trabalho; seção 3 é sobre a proposta da linguagem de especificação de intrusão; seção 4 apresenta os resultados obtidos da aplicação desenvolvida; seção 5 apresenta a conclusão do trabalho, e por fim, seção 6 apresenta os trabalhos futuros.

2. Trabalhos Relacionados

Este artigo foi baseado em alguns assuntos que serão descritos brevemente nessa seção: a seção 2.1 aborda sobre a computação em *grid*; 2.2 aborda sobre o Sistema de detecção de intrusão; 2.3 aborda as linguagens de especificação de intrusão conhecidas; 2.4 aborda sobre a linguagem *eXtensible Markup Language*.

1.1 *2.1 Computação em* grid

A nomenclatura *grid* (*Grid Computing*) é baseada nas *malhas* de interligação dos sistemas de energia elétrica. Em um sistema elétrico, o usuário utiliza a energia sem se perguntar aonde a mesma foi gerada. De maneira semelhante, a idéia de um *grid computacional* é que possamos criar um ambiente computacional distribuído que possua mecanismos que permitam o processamento, o armazenamento e o uso de equipamentos de forma transparente para os usuários da configuração [10].

Segundo a definição [17] um sistema distribuído é uma “coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente”. Sendo assim, a computação distribuída consiste em adicionar o poder computacional de diversos computadores interligados por uma rede de computadores trabalhando em conjunto no mesmo computador, para processar

colaborativamente uma determinada tarefa de forma coerente e transparente, ou seja, trabalhando como se fosse apenas um único e centralizado computador.

Uma boa vantagem sobre as outras tecnologias é o reaproveitamento das máquinas, evitando gastos excessivos para as organizações. Outro benefício que pode ser obtido através da computação em *grid* é o compartilhamento de dados entre organizações de pesquisa, além do próprio compartilhamento de ciclos de processamento. As organizações de pesquisa geralmente precisam de um enorme poder de processamento para realizar simulações ou análises de dados. Um sistema *cluster* resolveria o problema, porém utilizando a computação em *grid* não há necessidade de se utilizar computadores dedicados ao sistema, além de não ser preciso que eles estejam na mesma sala, cidade, estado ou país.

1.2 2.2 Sistemas de detecção de intrusão

Um IDS (*Intrusion Detection System*) é um sistema que detecta, de preferência em tempo real, violações de segurança em um sistema de informação e envia notificações de alerta ao gerente de segurança (pode ser tanto um ser humano ou uma máquina), o qual tem poder e direito de tomar as medidas mais apropriadas, como desconectar um usuário ou bloquear o tráfego de rede [16].

As técnicas para identificarmos os intrusos são geralmente classificadas em técnicas baseadas em anomalias e técnicas baseadas em assinaturas, cada um com suas vantagens e desvantagens.

Sistemas baseados em anomalias (comportamentos) aprendem o comportamento normal ou esperado dos usuários que utilizam o sistema e tentam identificar as divergências de uso. Se o uso está fora de um determinado limiar, uma notificação é disparada ao gerente de segurança.

Sistemas baseados em assinaturas (regras) ficam a procura de rastros ou padrões conhecidos resultantes do uso malicioso do sistema que ficam armazenados em um banco de dados de assinaturas de ataques. Dependendo da definição dessas assinaturas, esses sistemas podem até detectar ataques desconhecidos que são similares a alguns ataques conhecidos.

1.3 2.3 Detecção de intrusão baseado em regras

Essa técnica é a mais utilizada nos Sistemas de Detecção de Intrusão porque possui um baixo índice de alarmes falsos e um elevado índice de acertos. A limitação dessa técnica é que raramente consegue detectar padrões de ataques desconhecidos. Essa técnica é baseada em regras que monitoram um fluxo de eventos e agem como um algoritmo que procura as características maliciosas em um comportamento.

Utilizando um sistema especialista é possível descrever o comportamento malicioso em uma regra. As regras facilitam a evolução dos sistemas de detecção de intrusão porque uma nova regra pode ser adicionada sem mudar as regras existentes. Diferente dos sistemas baseados em anomalias (comportamentos) que toda vez que acrescenta uma nova característica é necessário realizar todo o aprendizado novamente (treinamento da rede neural).

Como é possível observar a produção de regras é o elemento chave desta técnica, pois é através dela que o sistema consegue reconhecer um ataque no ambiente. Elaborando um conjunto de regras boas pode assegurar uma boa segurança ao sistema. Vale lembrar que regras em excesso acabam restringindo a liberdade do usuário e acabam tornando o sistema inviável de ser utilizado.

Nas próximas subseções estão divididas da seguinte forma: a seção 2.3.1 contem informações sobre o Common Intrusion Specification Language e a seção 2.3.2 aborda sobre a linguagem de detecção de intrusão utilizada no Snort.

1.3.1 2.3.1 Common Intrusion Specification Language

O CISL (*Common Intrusion Specification Language*) foi desenvolvido a partir do CIDF (*Common Intrusion Detection Framework*). O CIDF tem por objetivo desenvolver protocolos e interfaces das aplicações para que os projetos na área de detecção de intrusão possam compartilhar informações e recursos. Tornando possível a reutilização dos componentes de detecção de intrusão em outros sistemas.

O número de pessoas ou máquinas que se passam por intrusos nos sistemas dos dias de hoje estão crescendo em grande escala. Em cada ambiente, a habilidade do IDS e o compartilhamento de informações com os seus componentes são muito importantes. Se um novo sistema possuir as informações de como os outros sistemas foram atacados, a probabilidade de que esse novo sistema seja atacado da mesma maneira diminui bastante, favorecendo os administradores de sistemas [12].

A linguagem CISL possui algumas limitações:

- Codificação muito limitada, só é possível descrever ataques de tamanho reduzido e com pouco tempo de duração;
- Muito antigo, documentação muito fraca. A documentação está na versão rascunho, a última atualização foi feita em 1999 e não possui uma lista dos verbos que a linguagem suporta. Provavelmente o projeto está parado;
- O CISL praticamente não se preocupa com as contas dos usuários;

- Linguagem muito fácil de se confundir devido a quantidade elevada de parênteses, como dito anteriormente, muito parecido com LISP [2].

1.3.2 2.3.2 Snort

O *SNORT* é uma ferramenta *NIDS* (Network Intrusion Detection System – “Sistemas de detecção de Invasão de Redes”) que detecta intrusão numa rede e é capaz de prevenir o sistema de ataques através de análise de tráfego na rede nos *IP*s da rede. O *SNORT* monitora o tráfego de pacotes em redes IP, realizando análises em tempo real sobre diversos protocolos (nível de rede e aplicação) e sobre o conteúdo [6].

Existe também, a possibilidade de utilizar métodos como o *Database Plug-in* por exemplo, para registrar pacotes em uma variedade de bases de dados diferentes (MySQL, PostgreSQL, entre outros), as quais contam com recursos próprios para efetuar consultas, correlações e dispõem de mecanismos de visualização para analisar dados.

Outro ponto positivo desse software é o grande número de possibilidades de tratamento dos alertas gerados. O subsistema de registro e alerta é selecionado em tempo de execução através de argumentos na linha de comando. Esses tratamentos são configurados através de regras que definem quando o sistema recebe um alerta, quais informações devem ser mantido em um *log*, quais informações devem ser desconsiderados e etc.

De acordo com [6] essas regras são descritas através de uma linguagem simples, flexível e bastante poderosa. A maioria das regras do SNORT são escritas em uma simples linha.

O *Snort* possui uma documentação muito boa e de fácil compreensão, mas não é possível aproveitá-lo para a computação em *grid*, devido ao fato de eles ser um NIDS e não um GIDS [16].

1.4 2.4 Extensible Markup Language

XML (extensible Markup Language) é uma linguagem de marcação recomendada pela *W3C* (World Wide Web Consortium) para gerar linguagens de marcação para necessidades especiais. É uma linguagem muito parecida com o *HTML* (Hyper Text Markup Language), a diferença é que o *XML* foi projetado para descrever os dados e dar ênfase em “o que” que aquele dado significa e o *HTML* é projetado para mostrar os dados e focar em como aquele dado deve aparecer.

O princípio do projeto era criar uma linguagem que pudesse ser ligada por software, e integrar-se com as demais linguagens. Sua filosofia seria incorporada por vários princípios importantes:

- Separação do conteúdo da formatação;
- Simplicidade de Legibilidade, tanto para humanos quanto para computadores;
- Criação de arquivos para validação da estrutura (DTD);
- Concentração na estrutura da informação e não na sua aparência.

3. Proposta

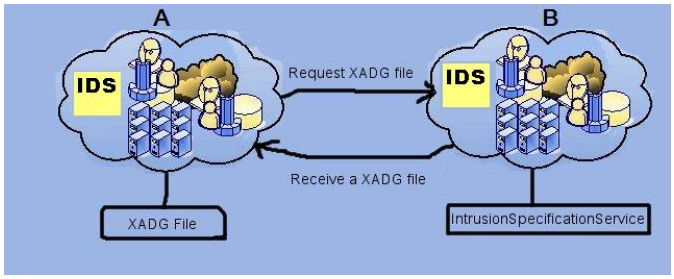
Conforme visto nas seções anteriores, as linguagens que especificam detecção de intrusão não possuem uma boa documentação [11] ou não são viáveis em ambientes *grid* [16]. A proposta do trabalho é definir a descrição de ataques em ambientes *grid,* utilizando a linguagem *XML* (devido a sua alta capacidade de integração com outros softwares, dentre outros motivos citado acima).

A proposta do trabalho se baseou em [18] que já utiliza a linguagem XML. Uma vantagem que se pode obter com o trabalho proposto é que padronizando a linguagem de detecção de intrusão, as atualizações das regras dos IDS podem ser feito de uma forma automática, por exemplo, através de um sistema similar ao *apt*[12].

Esta seção está dividido da seguinte forma: a seção 3.1 aborda sobre a arquitetura; 3.2 define o que é um ataque; 3.3 define a estrutura de um arquivo XADG; 3.4 mostra alguns exemplos de um arquivo XADG; 3.5 aborda sobre como usufruir melhor do arquivo XADG em um ambiente *grid*.

1.5 3.1 ARQUITETURA

A arquitetura do XADG (eXtensible Attack Definition in Grid) está definida na figura 3.1:



1.5.1.1

FIGURA 3.1 ARQUITETURA DO XADG DEMONSTRADO EM DOIS NODOS. NOTE QUE CADA NODO POSSUI O SEU PRÓPRIO IDS.

A figura 3.1 demonstra a arquitetura do XADG. Na figura demonstrada foi utilizado apenas dois nodos de um ambiente *grid* para facilitar o entendimento, mas poderia muito bem estar em um ambiente com dez nodos ou mais. Os passos a seguir descreve a arquitetura do XADG:

[12] Um programa utilizado por algumas distribuições do linux que busca nos repositórios as atualizações disponíveis para o sistema.

5. O administrador de um nodo, por exemplo A, possui um arquivo XADG muito antigo ou nem possui o mesmo e deseja obter esse arquivo;
6. O nodo pede para a *grid* o serviço de definição de ataque e recebe o nodo que possui o serviço desejado. O administrador faz uma requisição desse arquivo a esse outro nodo, por exemplo B;
7. Esse nodo retorna ao nodo solicitante o seu arquivo XADG;
8. Agora o nodo A analisa se deseja atualizar o arquivo (caso haja um desatualizado) e confirma a atualização.

1.6 3.2 DEFINIÇÃO DE ATAQUE

Por definição [18] ataque é qualquer atividade maliciosa direcionada a um sistema ou serviço, podendo ou não torná-lo indisponível. Alguns ataques sobre acesso ilegítimo e negação de acesso segundo [15] são: Engenharia social, explorar falhas no sistema (*/exploits/*), abuso de privilégios, mau uso, obtenção de privilégios e DoS.

1.7 3.3 ESTRUTURA DAS REGRAS DO XADG (eXtensible Attack Definition in Grid)

A proposta da estrutura do trabalho segue a estrutura das regras da linguagem utilizada por [18] com mais alguns campos, pois já está na linguagem XML. Os campos da estrutura de regras descrito são:

- *Alert:* Início de uma nova regra e possui o atributo *source* que representa a origem do dado. O atributo pode conter também os valores *log* e *message*;
- *Identification:* Representa um valor de identificação única da regra, análogo ao ID de um banco de dados;

- *RuleInformation:* Descreve a regra ou o evento intrusivo podendo informar os sistemas afetados;
- *Version:* Representa a versão da regra, auxiliando no controle das alterações;
- *Name:* Nome da regra;
- *Credits:* Autor ou a origem da regra;
- *Element:* Representa o elemento que pertence ao pacote de auditoria. Possui um atributo *value* que se refere ao nome do elemento e um valor para caracterizar a ocorrência do ataque. Uma regra pode ter mais de um *Element* para restringir a ocorrência;
- *Comprises:* Representa uma função lógica que proíbe um valor específico.
- *Range:* Representa uma função lógica que proíbe um intervalo de dados. Possui os atributos *MoreThan* e *LessThan* que define o intervalo superior e inferior, respectivamente.
- *InvokeLimit*: um valor que limite o número de referências ao elemento durante um certo tempo;
- *Interval*: complementando a regra acima, o intervalo de tempo da limitação.
- *Time*: o horário em que a regra é válida, por padrão, seria o dia todo. Mas também existe a possibilidade de limitar um horário em que se aplica a regra.

1.8 3.4 EXEMPLOS DE REGRAS

Na tabela 3.4.1 segue dois exemplos de regras utilizando a linguagem XADG:

<alert source=”communication”> <identification>rule1</identification> <ruleInformation> This rule verify the inappropriate use </ruleInformation> <version>1.0.0</version> <name>inappropriateUse</name> <credits>LRG</credits> <element value="service">web</element> <comprises>porn</comprises> </alert>	<alert source=”communication”> <identification>rule2</identification> <ruleInformation> This rule verify the DoS </ruleInformation> <version>1.0.0</version> <name>dos</name> <credits>LRG</credits> <element value="service">web</element> <invokelimit>10</invokelimit > <interval>60</interval> </alert>

TABELA 3.4.1 TABELA CONTENDO DUAS REGRAS: A) REGRA PARA EVITAR MAU USO E B) REGRA PARA EVITAR ATAQUE DE NEGAÇÃO DE SERVIÇO.

A regra representada pela tabela 3.4.1a é uma regra que atende o ataque descrito como **mau uso**. Uma regra que é necessária é que o elemento *service* possua o valor *web*, a regra não permite que ocorra a transmissão de qualquer texto contendo a palavra “porn”.

Na tabela 3.4.1b descreve uma regra que tenta evitar o ataque conhecido como **DoS**. A regra diz que o elemento *service* que possui o valor *web* não pode ser referenciado mais de 10 vezes, num limite de 60 segundos.

4. IMPLEMENTAÇÃO

Com o intuito de testar as funcionalidades do *IntrusionSpecificationService*, este capítulo apresenta o resultado da implementação do serviço. O serviço consiste em implementar a troca do arquivo XADG entre dois nodos, verificar as diferenças entre os dois arquivos (quando existir) e pedir a confirmação do administrador.

Esta seção se subdivide da seguinte forma: a seção 4.1 explica a escolha da plataforma escolhida para realização da implementação; a seção 4.2 mostra as principais partes do resultado do serviço; 4.3 aborda a validação do trabalho.

1.9 4.1 GRID-M

O *grid-m* é uma plataforma utilizada para construir aplicações de computação em *grid* e aplicações em dispositivos embargados e móveis. Essa plataforma fornece uma Application Programming Interface (API) para conectar aplicações desenvolvidas em Java em um ambiente de computação em *grid*. Seu perfil *runtime* é pequeno bastante para ser usado em aplicações de computação móvel. Acredita-se que as soluções do *grid-m* respondam à pergunta sobre uma plataforma para promover a homogeneização neste ambiente de desenvolvimento [14].

1.10 4.2 RESULTADOS

A plataforma escolhida para a validação do trabalho foi o *grid-m*, a principal justificativa para essa escolha é porque o produto foi desenvolvido por integrantes do laboratório LRG. Facilitando assim, o acesso ao código fonte, o contato com os desenvolvedores e utilização do XADG na plataforma.

Utilizando a plataforma *grid-m* a implementação do trabalho consistiu em criar a classe *IntrusionSpecificationService* que estende a classe *Service*. Essa classe realiza as trocas dos arquivos XADG entre dois nodos distintos que utilizam, no momento, a plataforma *grid-m*. Após realizar os trabalhos futuros, esse serviço pode realizar as trocas dos arquivos XADG de diferentes plataformas.

1.11 4.3 VALIDAÇÃO

A principal contribuição desse trabalho é a proposta do arquivo XADG que proporcionará uma segurança adicional ao ambiente *grid*. A implementação mostrada prova que com a padronização do arquivo que define um intruso em um ambiente *grid*, auxiliará bastante a vida de um administrador de um nodo, fazendo

com que ele possa se concentrar em outras questões de seguranças que ainda não são solucionáveis pelas máquinas (ou são solucionáveis de uma forma ineficiente).

5 CONCLUSÃO

Como em todo sistema de segurança é difícil prever a “criatividade” dos maliciosos e conseguir fazer um sistema de regras que além de prover uma boa segurança, ainda consiga tornar o ambiente agradável. Um bom exemplo é o sistema de alarme domiciliar.

Em relação a segurança, conseguimos agir de uma maneira mais eficiente se conhecemos um caso similar que já tenha acontecido. Podendo assim, tomar as medidas preventivas para que o mesmo caso não se repita.

Como a computação em *grid* ainda está em evolução. Ainda não existem tantos relatos em relação a mais novos tipos de ataques e também porque esse tipo de computação, no momento, só é utilizado por pessoas que são da área da computação.

Justamente essa falta de registros de ataques na computação em *grid* foi a maior dificuldade encontrada durante o trabalho para elaboração de regras que consigam prevenir os ataques mais sofisticados.

Mas com a realização dos trabalhos futuros, acredito que este trabalho consiga contribuir de uma forma positiva para a computação em *grid*. Ajudando na sua evolução em relação a segurança e que futuramente mais usuários a utilizem.

6 TRABALHOS FUTUROS

Os trabalhos futuros que podem ser citados são:

- Para facilitar a utilização do XADG, seria interessante manter paralelamente um *site*, análogo aos sistemas de *wiki* ou *blog* para facilitar a comunicação

entre elaboradores de regras XADG e integrá-lo ao *grid-m*. Pois havendo um compartilhamento entre mais elaboradores de regras, todos saem ganhando. O trabalho de conclusão de [8] aborda esse assunto;

- Deixar rodando o XADG em um ambiente *grid* que utiliza plataformas *grid* diferentes, juntamente com o IDS e registrar os ataques para que futuramente possa ser elaborada regras que consigam preveni-las e que tente manter a mesma eficiência;
- Implementar um serviço que elabore regras utilizando XADG baseado em logs de um IDS (utilizando alguma heurística).
- Testar compatibilidade do serviço com outras plataformas.

7. Referências

[1] ALLEN, Julia et al. *State of the Practice of Intrusion Detection Technologies*. Technical Report CMU/SEI-99-TR-028, Software Engineering Institute, Carnie Mellon University, jan. 2000.

[2] *Association of Lisp Users*. Disponível em: http://lisp.org/alu/home. Acesso em: 20 nov. 2007.

[3] AXELSSON, Stefan. *Research in Intrusion-Detection Systems: A Survey*. Technical

Report TR-98-17, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, aug. 1999. 93 p.

[4] BARBOSA, André. *Sistemas de Detecção de Intrusão*. Rio de Janeiro: COPPE/UFRJ, 2000.

[5] BUYYA, Rajkumar. *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. 2002. Melbourne, Australia: Monash University, 2002.

[6] *Chapter 2 Writing snort rules. How to write snort rules and keep your sanity*. Disponível em: http://www.snort.org/docs/writing_rules/chap2.html. Acesso em: 5 out. 2007.

[7] CHRYSSOVERGIS, Marcelo Digiacomo. *Um componente para verificação de políticas de segurança no Globus Toolkit 3*. Florianópolis: UFSC, 2006.

[8] CONTE, Ezequiel. *Intranet na CERTI: Um modelo baseado em recursos de Enterprise 2.0*. Florianópolis: UFSC, 2007.

[9] CORNELLI, F., et al. *Choosing Reputable Servents in a P2P Network*, In: Eleventh International World Wide Web Conference, Honolulu, Hawaii, pp. 376-386, 2002.

[10] DANTAS, Mario. *Computação Distribuída de alto desempenho: Redes, clusters e grids computacionais.* Axcel Books, 2005.

[11] DOYLE, Jon. *Some Representational Limitations of the Common Intrusion Specification Language*. Cambridge: MIT, 1999.

[12] FEIERTAG, R. A Common Intrusion Specification Language (CISL). Disponível em: http://gost.isi.edu/cidf/drafts/language.txt. Acesso em: 10 abr. 2007.

[13] FOSTER, I. *Internet Computing and the Emerging Grid*. *Nature Web Matters*, 2000. Disponível em: http://www.nature.com/nature/webmatters/grid/grid.html. Acesso em: 11 fev. 2007.

[14] FRANKE, Hans. *Grid-m: Middleware para integrar dispositivos móveis, sensores e computação em grid*. Florianópolis: UFSC, 2007.

[15] KENDALL, Kristopher. *A Database of Computer Attacks for the Evalution Intrusion Detection Systems*. Cambridge: MIT, 1999.

[16] SCHULTER, Alexandre. *Integração de Sistemas de Detecção de Intrusão para Segurança de Grids Computacionais*. Florianópolis: UFSC, 2006.

[17] TANENBAUM, Andrew S. *Distributed System: Principles and Paradigms*. 1. st. edition. **Prentice Hall**, 2002.

[18] VIEIRA, Kleber. *Mecanismos para aplicação paralela de técnicas distintas de detecção de intrusão em ambientes de grid*. Florianópolis: UFSC, 2007.

[19] *Xml Tutorial*. Disponível em: http://www.w3schools.com/xml/default.asp. Acesso em: 05 out. 2007.