FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

PEDRO VINÍCIUS PEREIRA

IDS, UMA FORMA DE PREVENÇÃO DE ATAQUES – O IDS STAT

MARÍLIA
2008

PEDRO VINÍCIUS PEREIRA

IDS, UMA FORMA DE PREVENÇÃO DE ATAQUES – O IDS STAT

Monografia apresentada ao Curso de Graduação em Ciência da Computação, do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, como requisito para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora:
Profa. Dra. KALINKA R. L. J. CASTELO BRANCO

MARÍLIA
2008

PEDRO VINÍCIUS PEREIRA

IDS, UMA FORMA DE PREVENÇÃO DE ATAQUES – O IDS STAT

Banca Examinadora da monografia apresentada ao Curso de Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Grau de Bacharel em Ciência da Computação.

Resultado: 5,0 (Cinco)

ORIENTADOR: _____
Profa. Dra. Kalinka R. L. J. Castelo Branco

1º EXAMINADOR: _____
Prof. Dr. Ildeberto Aparecido Rodello

2º EXAMINADOR: _____
Profa. Juliana de Oliveira

Marília, 10 de novembro de 2008.

Dedicatória

Dedico este trabalho primeiramente à Deus, por mais esta vitória em minha vida.

Dedico-o também a minha família e minha namorada Milena que sempre estiveram ao meu lado, em todos os momentos.

Agradecimentos

Agradeço a profa. Dra. Kalinka Regina L. J. Castelo Branco pela orientação que foi imprescindível para a conclusão deste trabalho.

Agradeço também aos amigos que de alguma forma me ajudaram na conclusão deste trabalho, sendo desde uma ajuda mais especifica, como também com palavras de incentivo para nunca desistir de meus objetivos.

PEREIRA, Pedro Vinícius. **IDS, uma forma de prevenção de ataques – O IDS STAT.** 2008. 52f. Monografia (Bacharelado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2008.

RESUMO

Com o crescimento do uso das redes de computadores e da *Internet*, a segurança é algo sempre requisitado pelas empresas a fim de estarem sempre protegidas contra ataques. Porém, com o crescimento do número de ataques, algo mais específico para a proteção das empresas precisou ser criado, então surgiu a idéia dos Sistemas de Detecção de Intrusão (IDS). Os IDS têm como objetivo detectar intrusões antes que as mesmas consigam obter sucesso e vêm sendo cada vez mais utilizados como uma forma de prevenir ataques a redes de computadores. Esta monografia tem como objetivo especificar um IDS e apresentar o IDS STAT, uma ferramenta nova, mas que possui grandes chances de crescimento no meio dos IDS, por ser gratuita e possuir uma metodologia teoricamente simples para implementação de assinaturas de ataques contra intrusões. O estudo inicia com a apresentação de como funciona um IDS, seguido dos principais tipos de ataques atualmente existentes, fechando com a apresentação do STAT e da sua linguagem para especificação de um cenário de ataque.

Palavras-chave: Sistema de Detecção de Intrusão, STAT, segurança.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AlertSTAT – *Alert State Transition Analysis Technique*

BSM – *Basic Security Module*

CPU – *Central Processing Unit*

DDoS – *Distributed Denial of Service*

DNS – *Domain Name Service*

DoS – *Denial of Service*

HTTP – *Hiper Text Tranfer Protocol*

ICMC – Instituto de Ciências Matemáticas e de Computação

IDMEF – *Intrusion Detection Exchange Message Format*

IDS – *Intrusion Detection System*

IP – *Internet Protocol*

NetSTAT – *Network State Transition Analysis Technique*

NSTAT – *N State Transition Analysis Technique*

SMTP – *Simple Mail Transfer Protocol*

STAT – *State Transition Analysis Technique*

STATL – *State Transition Analysis Technique Language*

TCP – *Transmission Control Protocol*

TFN – *Tribe Flood Nerwork*

UDP – *User Datagram Protocol*

USTAT – *UNIX State Transition Analysis Technique*

WebSTAT – *Web State Transition Analysis Technique*

WinSTAT – *Windows State Transition Analysis Technique*

SUMÁRIO

INTRODUÇÃO

Atualmente com a grande automatização das empresas e com o crescimento do uso da *Internet*, é muito importante que as informações estejam sempre protegidas e não possam ser interceptadas. Logo, o monitoramento das redes de computadores é essencial para evitar possíveis problemas como ataques e vírus. Com base nisso, a procura por sistemas detectores de intrusão cresce a cada dia (JUNIOR, 2003).

A segurança é sempre um grande problema para os administradores de redes. Como fazer para estar protegido e sempre atualizado com relação às novas técnicas de ataque? Normalmente são utilizadas técnicas e ferramentas que permitem a monitoração e detecção de ataques. A monitoração contra ataques e intrusões é fundamental na estrutura de segurança de uma rede de computadores, pois auxilia o administrador da rede a prevenir ataques e a agir quando um ataque é iniciado ou detectado.

Sistemas que detectam e bloqueiam os ataques antes que eles sejam concluídos com sucesso são chamados de Sistemas de Detecção de Intrusão (IDS – *Intrusion Detection Systems*). O IDS é visto como a evolução do conceito de *firewall* (JUNIOR, 2003), logo, como mais uma ferramenta para proteção das informações de uma empresa e que vem permitindo um crescimento na sua utilização devido a grande procura de ferramentas cada vez mais potentes na ajuda para evitar intrusões.

Este trabalho descreve o que são IDS e os possíveis tipos de ataques a uma rede de computadores, e tem como objetivo principal o estudo do IDS STAT.

Assim a organização desta monografia segue a ordem apresentada a seguir:

- Capítulo 2 descreve o que é sistema de detecção de intrusão e apresenta uma comparação entre dois IDS.
- Capítulo 3 aborda os principais tipos de ataques a uma rede de computadores e o problema causado por cada um desses ataques.
- Capítulos 4 e 5 apresentam e detalham a ferramenta STAT, que será utilizada para geração de assinaturas. São descritos os conceitos bem como os métodos para geração de assinaturas.
- E por fim são descritas as conclusões.

CAPÍTULO 2 - SISTEMA DE DETECÇÃO DE INTRUSÃO

Detecção de intrusão vem a ser o ato de identificar quaisquer alterações no sistema a fim de procurar identificar se um ataque está iniciando ou ocorrendo. Desse modo, "a detecção de intrusão da rede permite identificar e reagir a ameaças contra o seu ambiente" (NORTHCUTT, 2002).

O Sistema de Detecção de Intrusão (IDS) é uma ferramenta de gerenciamento de segurança que auxilia na monitoração dos eventos ocorridos em uma rede, detectando qualquer comportamento que não seja considerado normal.

As principais características de um IDS segundo Balasubramaniyan *et al.* (1998) são:

- Executar continuamente sem interação humana;
- Ser seguro o suficiente de forma a permitir sua operação em *background* (segundo plano);
- Ser tolerante a falhas de forma a não ser afetado por uma queda do sistema (seja essa acidental ou por ações humanas);
- Resistir às tentativas de subversão (mudança), ou seja, deve monitorar a si próprio de forma a garantir sua segurança;
- Ter o mínimo de impacto no funcionamento do sistema, ou seja, gerar uma sobrecarga mínima no sistema computacional que está sendo executado;
- Poder detectar mudanças no funcionamento normal;
- Permitir fácil configuração: cada sistema possui padrões diferentes de modo que o IDS deve ser adaptado de forma fácil a esses diversos padrões (de acordo com a política de segurança);
- Adaptar-se às mudanças no sistema e ao comportamento dos usuários;
- Ser escalável para dar suporte à carga, de monitorar várias estações e ainda produzir resultados confiáveis em tempo hábil para a tomada de decisões;
- Ser difícil de ser enganado (fraudado).

Muitos IDS funcionam a partir da análise de padrões do sistema operacional e da rede, tais como, utilização de *login*, uso da memória, uso da CPU, entre outros, a fim de

estabelecer uma base de informações. A partir dessas informações o IDS pode detectar uma possível tentativa de intrusão e até identificar o método de invasão utilizado.

Os IDS vêm ocupando um lugar de destaque na proteção de redes, mas não devem ser usados sozinhos como uma fonte de segurança da rede, nem serem utilizados como substitutos de um *firewall*, mas devem ser somados a outros métodos para permitir uma melhora na segurança da rede.

Um IDS é composto basicamente por dois dispositivos principais, o Console de Comando e o Sensor. O Console de Comando, ou simplesmente Console, tem a função de controlar o IDS, monitorando o estado do sensor e processando alertas enviados pelo sensor (PROCTOR, 2001). "O sensor é o dispositivo responsável pela coleta de informação para análise de descoberta de uma invasão" (CROTHERS, 2003).

2.1 Abordagem do Problema de Detectar Intrusões

Existem diferentes classificações de IDS, no entanto as principais são: o baseado em Rede e o baseado em *Host*.

2.1.1 Baseado em Rede

Os IDSs baseados em rede são, geralmente, compostos por sensores que são responsáveis por analisar os pacotes que trafegam pela rede e pela estação de console que gera os alertas.

Stephen Northcutt (NORTHCUTT, 2002, p. 156) possui a seguinte definição:

> "O sensor de IDS da rede fareja o tráfego e o analisa, procurando várias assinaturas que poderiam indicar um *scan* ou uma sonda, atividade de reconhecimento ou tentativa de explorar uma vulnerabilidade. Tradicionalmente, os sistemas de detecção de intrusão não interferem com o tráfego de alguma forma. Ao contrário de um *firewall* ou filtro de pacotes, que tomam decisões sobre qual tráfego permitir, um sensor de detecção é, na realidade, um farejador de pacotes que também realiza análise. Contudo alguns sensores estão começando a dar respostas ativas para o tráfego suspeito, como ao terminar conexões TCP suspeitas."

Os sensores são espalhados pela rede capturando todos os pacotes que circulam pelo segmento de rede, independente de qual for o seu destino. Geralmente são dispositivos passivos, não causando muito impacto no desempenho da rede.

Os sensores não interferem no tráfego, somente analisam o que esta trafegando pela rede, não interferindo na origem e destino dos pacotes enviados. São invisíveis para muitos invasores, pois como não interferem no tráfego dos pacotes na rede fica difícil do invasor detectar a presença do sensor. É possível instalar vários sensores em um segmento de rede, como vantagem se tem a tolerância à falhas, sendo que na maioria das vezes essa opção não é utilizada devido ao seu custo ser elevado.

Os IDS baseados em rede possuem algumas dificuldades no monitoramento de redes de grande porte que possuem tráfego intenso, pois devido a esse tráfego os sensores podem não ser rápidos o suficiente para monitorar tudo o que esta passando pela rede. Ademais, possuem dificuldades em monitorar uma rede ligada por *switch*, uma vez que o *switch* cria uma conexão direta entre a origem e o destino, não enviando os pacotes para todo o segmento de rede, assim o sensor não captura tudo o que está passando pela rede. Também não são capazes de analisar pacotes criptografados, pois o IDS não reconhece a forma que o pacote está escrito. E possuem problemas com pacotes fragmentados, pois os mesmo não conseguem ser montados pelo IDS.

2.1.2 Baseado em *Host*

Os IDS baseados em *host* monitoram máquinas individuais, permitindo uma melhor análise (comparado ao IDS baseado em rede) e gerando menos falso positivo.

Podem trabalhar de três formas (NORTHCUTT, 2002):

- Verificando a integridade do sistema de arquivos: procuram por mudanças não autorizadas no sistema de arquivos a partir de uma base criada do sistema quando considerado confiável. É configurável quando é possível indicar quais arquivos ou diretórios podem sofrer alterações, diminuindo assim os alertas;
- Verificando a conexão da rede: verifica as conexões do *host* a procura de ataques ou atividades maliciosas. Tem menos problemas com a sobrecarga de tráfego, pois monitora somente o tráfego destinado a um determinado *host*;
- Verificando os arquivos de *log*: observam o conteúdo dos *logs* e avisam quando algo suspeito é detectado. Possui uma vantagem, se vários *hosts* salvarem seus *logs* em um único ponto este sistema pode monitorar mais do que um *host.*

Os IDS baseados em *host* conseguem monitorar eventos locais, como por exemplo, alterações em arquivos, enquanto que o IDS baseado em rede não consegue. Conseguem

também trabalhar em redes com criptografias, pois como analisam o *host*, verificam os arquivos antes e depois de serem criptografados. Não possuem problemas em redes com *switch* uma vez que analisam o conteúdo que entra e sai do *host* onde está instalado, possibilitando a detecção de cavalos de tróia e outros ataques que envolvam brecha de *software*.

Entretanto, esses sistemas são mais difíceis de gerenciar uma vez que as informações precisam ser configuradas e gerenciadas para cada estação monitorada. Eles não podem detectar invasões direcionadas a toda a rede. O *host* pode sofrer um ataque, o IDS não percebe e por fim pode ser comprometido ou desabilitado.

Ao contrário dos IDS baseados em rede, o IDS baseado em *host* influencia no desempenho do *host* no qual está instalado uma vez que consome recursos para o processamento das informações.

2.2 Quanto ao Método de Análise

Alertas são gerados devido a um dos dois tipos de métodos de detecção: Baseado em Assinaturas e Baseado em Anomalia.

2.2.1 Baseado em Assinaturas

Este método utiliza regras pré-definidas no tráfego da rede à procura de algum ataque conhecido. Quando é encontrado algum código na rede que pertence a alguma regra um alerta é gerado ou algum evento de ação defensiva é gerado. Essas regras pré-definidas são chamadas assinaturas. Exemplos de sistemas de detecção baseados em assinatura o IDIOT (KUMAR E SPAFFORD, 1995), o STAT (ILGUN et al., 1995) e o SDI que foi desenvolvido no ICMC (Instituto de Ciências Matemáticas e de Computação) da USP de São Carlos (CANSIAN, 1997).

As assinaturas são desenvolvidas pelo desenvolvedor do IDS ou pelo administrador da rede, como também podem ser encontradas em sites especializados em segurança, neste caso, pode ser necessário adaptar a regra para cada IDS especifico.

Para se criar uma regra ou filtro, é necessário ter os seguintes conhecimentos (NORTHCUT, 2000):

- A linguagem para escrever o filtro;

- A rotina de instalação do filtro;
- A assinatura do ataque para o qual o filtro está sendo escrito;
- Conhecimento sobre a arquitetura de redes TCP/IP;

É necessário certo cuidado ao criar uma nova regra, pois o menor erro pode causar o comprometimento da detecção do IDS. Para evitar erros, o mais aconselhável é testar as regras depois de criá-las.

2.2.2 Baseado em Anomalias

Possui uma base de dados do comportamento padrão do sistema, a partir dessa base, o sistema detecta o que é ou não permitido e quando encontra algo fora do padrão gera um alerta. Os IDSs baseados nesse tipo de método são mais complicados de configurar, pois é mais difícil estabelecer o que é padrão em uma rede com muitos usuários (STANGER, 2002).

Uma vantagem deste método é que ele pode detectar ataques não conhecido, pois não funciona analisando uma regra específica. Como desvantagem tem-se um alto número de falso-positivos [1].

2.3 Sistemas de Detecção de Intrusão

Existem na literatura diferentes sistemas de detecção de intrusão. Dentre os IDSs baseados em assinaturas destacam-se o SNORT (SNORT, 2008) e o STAT (ILGUN et al, 1997). O SNORT é um IDS amplamente utilizado, entretanto, a aquisição das suas assinaturas é feita mediante pagamento. O STAT é um IDS aberto e gratuito, e que possui um modo de obtenção de assinaturas baseado em máquinas de estado.

2.3.1 SNORT

O SNORT é um sistema de detecção de intrusão baseado em rede amplamente utilizado, tendo sido desenvolvido por Marty Roesch em 1998. Possui versões para vários sistemas operacionais, entre os quais se pode citar *Linux, OpenBSD, FreeBSD, NetBSD, Solaris, SunOS* 4.1.x, *Windows*, entre outros (SNORT, 2008).

[1] Quando um sensor detecta uma atividade normal como sendo um ataque. Para exemplificar tem-se que um sistema tem em sua base de dados que um usuário sempre utilizou a *Internet* pela parte da manhã, caso o usuário a utilize na parte da noite, um alerta será gerado.

O SNORT possui uma arquitetura simples baseada em *plugins*, executando basicamente as funções de captura de pacotes na rede, análise dos pacotes e geração de alertas. É um sistema leve, capaz de trabalhar em grandes redes e detectar uma grande variedade de ataques em tempo real, sendo o seu sistema de detecção baseado em assinaturas.

O SNORT pode ser configurado para trabalhar de três modos:

- *Sniffer*: simplesmente lê os pacotes da rede e mostra o resultado no console do programa gerenciador;
- Gerador de *log* de pacotes: trabalha de maneira semelhante ao modo *sniffer*, porém armazena todos os pacotes em arquivo para uma análise futura;
- Detector de intrusão: é o método mais flexível e completo para analisar o tráfego da rede. Podem-se definir novas regras de detecção além das já disponíveis. Neste modo, somente são gerados alertas ou armazenados no *log* os pacotes definidos nas regras.

O SNORT possui a facilidade de além de incorporar as regras disponíveis na *Internet*, permitir a criação de novas regras. É comparando o tráfego do segmento de rede com as regras existentes que o SNORT detecta o código malicioso e gera o alerta ou toma medidas de contra-ataque. Entretanto, apesar de bastante utilizado o SNORT é um IDS pago, o que inviabiliza o seu estudo no contexto deste trabalho.

2.3.2 STAT

O STAT explora o uso da análise de transição de estados para descobrir a intrusão em tempo real (STAT, 2008).

A análise no STAT é feita por meio da filtragem e abstração das informações. Estas abstrações, que são mais adequados para análise, portabilidade e compreensão humana, são chamadas de assinaturas. As ações de assinatura movem o sistema para uma seqüência de estados, cada mudança de estado deixa o sistema perto de uma configuração acertada. Seqüências de intrusão são definidas pelos diversos estados que são capturados em um sistema baseado em regras.

Mais detalhes a respeito do STAT serão dados no capitulo 4, uma vez que esta ferramenta é alvo de estudo deste projeto.

2.4 Considerações Finais

Este capítulo abordou conceitos, funcionalidades e formas de um Sistema de Detecção de Intrusão, elencando duas ferramentas de Detecção de Intrusão: o STAT e o SNORT.

Ataques em redes de computadores têm acontecido cada vez com mais freqüência. Ferramentas para Detecção de Intrusão têm sido bastante estudadas com o objetivo de elaborar novas formas de evitar esses ataques. Entretanto, para que essas ferramentas possam ser eficazes, há a necessidade de se conhecer mais a fundo os tipos de ataque que podem ocorrer, desse modo, o capítulo seguinte aborda as ameaças mais comuns às redes de computadores.

CAPÍTULO 3 - PRINCIPAIS TIPOS DE ATAQUES À REDE DE COMPUTADORES

Primeiramente cabe proferir a definição do que vem a ser um ataque diferenciando-o do que vem a ser uma intrusão. Um ataque é uma tentativa de intrusão. Uma intrusão é um ataque que cumpriu com seus objetivos (CROTHERS, 2003).

Segundo Kurose e Ross (2005) os principais tipos de ataques à rede de computadores são: vírus, *worms*, cavalos de tróia, *port scanning* (varredura de portas), DoS (*Denial of Service* – negação de serviço), DDoS (*Distributed Denial of Service* – negação de serviço distribuído). Esses ataques são descritos nas seções que seguem.

3.1 Vírus, *Worms* e Cavalos de Tróia

Vírus, *worms* (vermes) e cavalos de tróia (*trojans horses*) são típicos exemplos de códigos maliciosos e constituem grandes ameaças para as redes.

3.1.1 Vírus

Os vírus são programas de computadores (ou fragmentos de programas), geralmente maliciosos, que se propagam infectando o computador, gerando cópias de si mesmo e tornando-se parte de outros programas de computador. Eles dependem da execução do programa hospedeiro para serem ativados e continuarem o processo de infecção. Além de serem capazes de se reproduzirem, eles podem também corromper arquivos e sistemas (CARTILHA, 2008).

A propagação dos vírus se dá por meio de pen-drives, CD-ROM, documentos infectados que são executados, de e-mails (com anexos infectados), de programas piratas, da execução de *downloads* de procedência duvidosa, entre outros. Em todos esses exemplos, faz-se necessária a presença e atuação do ser humano.

Alguns vírus são pré-programados para danificar o computador de forma a corromper os programas, excluir arquivos ou até mesmo simplesmente construídos para transmitir mensagens cujo intuito é chamar a atenção de um número grande de pessoas. Independentemente do intuito, constituem, normalmente, situações que acabam por culminar na queda ou parada temporária do sistema.

3.1.2 *Worms*

Os *worms* podem ser considerados uma espécie de vírus, mas diferentemente do vírus, os *worms* não precisam da interação do usuário para se espalhar. Logo são considerados mais perigosos que os vírus normais (MOURA, 1999).

Os *worms* são programas capazes de se reproduzirem de um computador para o outro, assim como os vírus, mas, ao contrário dos vírus, os *worms* não utilizam um programa como hospedeiro. Eles possuem a capacidade de se propagarem automaticamente no próprio computador ou de computador para computador, realizando assim a infecção.

Um exemplo de propagação de *worms* é o do ***Melissa*** (CARTILHA 2008), que após infectar um computador ele mesmo procura os endereços no programa cliente de *e-mail* da vítima e transmite, de forma transparente para o usuário, o documento infectado para todos os *e-mails* cadastrados no contato da vítima.

3.1.3 Cavalos de Tróia

O termo vem de uma passagem da mitologia grega de Ilíada e Homero, na qual os gregos deram de presente um imenso cavalo de madeira a seus inimigos, os troianos, aparentemente como oferta de uma proposta de paz. Porém, após os troianos terem arrastado o cavalo para dentro das paredes da cidade, soldados gregos que estavam escondidos na barriga oca do cavalo saíram à noite e abriram as portas da cidade permitindo a seus compatriotas invadir e capturar a cidade.

Por analogia, na informática, o termo *trojan* ou cavalo de tróia é usado para designar uma categoria de programas destrutivos mascarados em programas e aplicativos benignos (CARTILHA, 2006).

Os *trojans* possuem muitas características similares aos vírus, tais como: perda de arquivos, falhas na memória, erros em periféricos, entre outros. A grande diferença é que o *trojan* pode ser considerado um vírus inteligente, pois é controlado à distância pela pessoa que o instalou. Esse indivíduo consegue “enxergar” o computador atacado, podendo realizar desde as mais simples tarefas como mexer o mouse e chegar até a utilização do seu IP como ponte para outros ataques. Conseguem ficar escondidos em arquivos de inicialização do sistema operacional e se iniciam toda vez que a máquina é ligada.

A popularização da Internet e a facilidade de se criar um programa cavalo de tróia fazem com que esse método de invasão seja atualmente o mais perigoso de todos. Ele não depende de falhas no seu sistema, e é quase indetectável. Pode-se esconder um *trojan* em fotos, arquivos de música, aplicativos e jogos (CARTILHA, 2006).

3.2 *Port Scanning* - Varredura de Portas

Port scanning é uma técnica de varredura de portas TCP (*Transmission Control Protocol*) comum a *crackers* para reconhecimento de sistemas alvo. Esse ataque consiste em testar as portas de um *host*, ou mesmo de um grupo de *hosts*, a fim de determinar quais dessas portas estão em condições de aceitar conexões. Dessa forma, um programa de *port scan* é aquele que averigua conexões cujos números de porta são bem conhecidos para detectar informações e serviços em execução no sistema alvo. Assim, com base na lista de portas que estão aguardando conexões, o *cracker* pode escolher entre um ou outro método de invasão (KUROSE e ROSS, 2005).

O *port scanning*, eventualmente, também é capaz de revelar outras informações de interesse do atacante tais como o sistema operacional em execução, a partir da análise de como o alvo reage aos eventos gerados durante a varredura.

Podem ocorrer ligeiras diferenças na forma de como é realizado o *port scanning*, mas ele consiste, basicamente, em enviar uma série de requisições, seja via TCP ou UDP (*User Datagram Protocol*), para um *range* de portas de um determinado *host* e verificar as respostas. Essas respostas podem ser utilizadas para determinar quais serviços estão ativos, por exemplo: *Web* (*HTTP - Hiper Text Transfer Protocol* – porta 80), transferência remota de arquivos (*FTP - File Transfer Protocol* – porta 21), ou *e-mail* (*SMTP – Simple Mail Tranfer Protocol* – porta 25) (KUROSE E ROSS, 2005).

3.3 DoS (*Denial of Service*) - Negação de Serviço

Outra categoria de ataque que constitui uma ameaça à segurança das redes de computadores é o classificado ataque de negação de serviço.

Trata-se de um tipo de ataque que não necessita de acesso ou invasão à rede-alvo, mas pode acarretar sérios transtornos dependendo da criticidade do ambiente envolvido. Tem como finalidade, tornar impossível a utilização dos recursos de uma rede ou de um

determinado host. Constitui em ataques baseados na sobrecarga da capacidade ou em uma falha não prevista (KUROSE E ROSS, 2005).

Esse tipo de ataque tem como objetivo esgotar os recursos do sistema alvo, forçando uma interrupção total ou parcial dos serviços. A capacidade de processamento, de armazenamento de dados e a largura de banda são alguns dos recursos visados pelas técnicas de negação de serviços.

O problema principal deste tipo de ataque está focado no protocolo IP, que é altamente vulnerável a ataques DoS. Além disso, muitas ferramentas de ataques estão disponíveis para o acesso público e são relativamente fáceis de utilizar.

Embora existam técnicas de negação de serviços destinadas a atacar *hosts* e computadores pessoais, a maioria dos ataques costuma ser direcionada contra sistemas de maior porte, os quais oferecem serviços a um grande número de usuários, como é o caso de servidores *WEB*, ou que desempenham funções críticas para o funcionamento de redes e sistemas distribuídos.

Outros motivos para existirem esse tipo de falha nos sistemas é um erro básico de programadores, falhas na implementação e *bugs* (erros), além de outras peculiaridades dos sistemas operacionais, na medida em que podem oferecer oportunidades para comprometer o funcionamento do sistema (pela incapacidade de tratar erros). Desse modo, o invasor inicia da premissa de que erros existem e que deve ficar efetuando diversos tipos de testes de falhas, até acontecer um erro e o sistema parar sua execução.

Esse tipo de ataque não causa perda ou roubo de informações, mas é um ataque preocupante, pois os serviços do sistema atacado ficarão indisponíveis por tempo indeterminado. Cabe ainda salientar que quando um computador/*site* sofre ataque DoS, ele não é invadido, mas sim sobrecarregado. Isso independentemente do sistema operacional utilizado.

Um ataque DoS pode ainda ser visto como um *worm* que se prolifera entre servidores infectados procurando novos computadores para se proliferar, entretanto, como o programa não se autodetecta ele se re-instala consumindo recurso das máquinas já infectadas exaurindo assim os recursos das máquinas.

Ataques de DoS podem ser lançados contra roteadores de borda, *bastion hosts*, e *firewalls*, sendo que roteadores, servidores DNS (*Domain Name Service*) e *firewalls* costumam ser os alvos preferenciais (KUROSE E ROSE, 2005). Em ataques direcionados aos equipamentos de redes, o objetivo principal é tirar uma rede ou sub-rede inteira do ar e não somente algumas máquinas específicas. Existem *bugs* em vários tipos de roteadores e em

outros equipamentos de conectividade que podem facilitar esses ataques. Ataques aos *firewalls* fazem com que esses percam a função de filtro, deixando passar conexões TCP para qualquer porta, sendo que um ataque DoS pode, e normalmente é, utilizado para facilitar uma invasão.

É importante ainda salientar que as técnicas de negação de serviços são freqüentemente adotadas como uma etapa intermediária de métodos de ataque mais complexos. Dessa maneira, elas servem como uma armadilha para deixar um *host* (sistema ou servidor) fora do ar a fim de que outro *host* assuma sua identidade ou, até mesmo, interrompa o funcionamento de um sistema que execute funções de segurança e controle da rede.

3.4 DDoS (*Distributed Denial of Service*) - Negação de Serviço Distribuído

Ataques DDoS constituem sérios problemas que afetam os usuários de *Internet*, uma vez que consomem os recursos de um *host* ligado à rede que prestam serviços, tais como *e-mail* (SMTP) e páginas *Web* (HTTP) (NORTHCUTT, 2000).

Basicamente, os ataques DDoS são coordenados por um atacante que de posse de *hosts* dedicados, conhecidos como *zumbis*, lança um ataque coordenado sobre uma rede ou *host* denominado vítima (KUROSE E ROSS, 2005). Esse tipo de ataque conquistou fama no início do ano 2000, devido ao aumento do uso da *Internet*. A Figura 1 ilustra um exemplo de ataque DDoS.

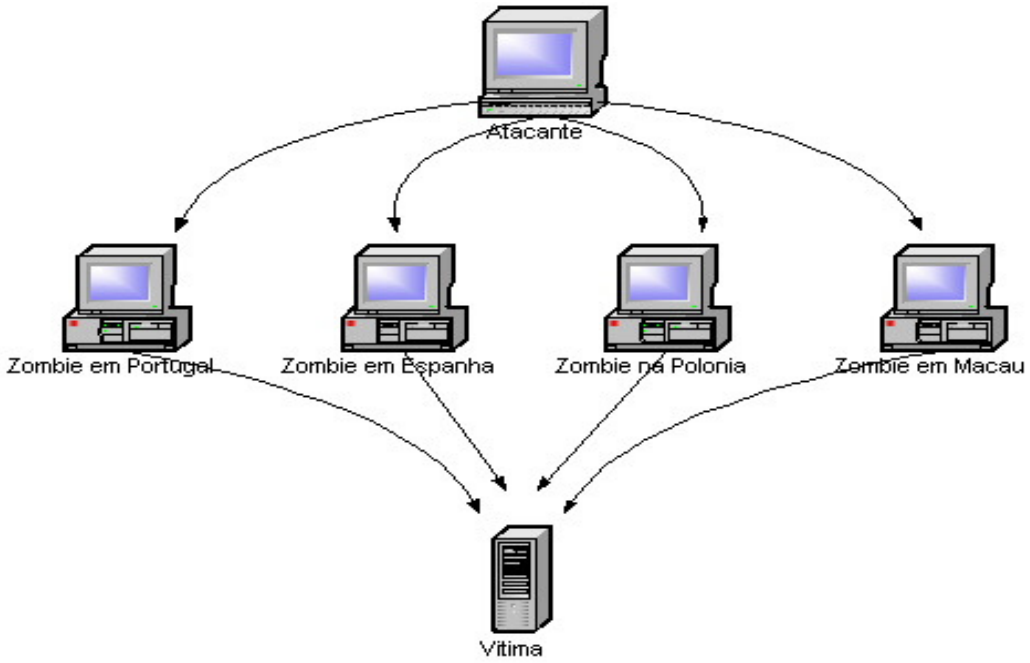

Figura 1. Exemplo de ataque DDoS (B.I.S.S, 2008)

O DoS realiza o ataque por meio de um único computador. Mas, com o passar dos tempos observou-se que a idéia poderia ser expandida, utilizando uma série de computadores atacantes ao mesmo tempo poder-se-ia obter resultados mais desastrosos e eficientes, levado ao DDoS. Em verdade o ataque DDoS potencializa os danos causados pelos ataques de negação de serviço.

Para que o ataque consiga sucesso faz-se necessário que sejam empregados *software*s específicos que visam organizar esse ataque. Tem-se como exemplo dessas ferramentas o *TFN* (*Tribe Flood Network*) (DITTRICH, 2008), o *Stacheldrah* (DITTRICH, 2008) e o *Trinoo* (DITTRICH, 2008). Essas ferramentas são instaladas em alguns *hosts* que atuarão como servidores (mestres). Paralelamente, outros *hosts* recebem também componentes de *software*, passando por sua vez a representar o papel de clientes (escravos).

As instalações tanto dos servidores quanto dos módulos clientes são feitas de forma não autorizada, ou seja, os componentes são embutidos em outros programas supostamente inofensivos. Ao comando do atacante, os servidores se comunicam com os clientes, determinando o início do ataque, seguidos pelos *hosts* que executam o módulo cliente que lançam ao mesmo tempo uma série de ataques contra o alvo ou os alvos especificados.

Tanto para efetuar como para tentar evitar um ataque DDoS fazem-se necessárias ferramentas com um alto nível de sofisticação, integrando recursos avançados que vão desde mecanismos de distribuição automatizada dos módulos clientes até comunicações criptografadas entre os servidores e os clientes.

Dessa forma, os ataques DDoS merecem especial atenção, não apenas pela eficácia, mas também por estabelecer um novo modelo de ataque distribuído.

3.5 Considerações Finais

Este capítulo abordou alguns dos tipos mais comuns de ataques às redes de computadores.

Tendo em vista o crescente aumento de ataques, faz-se necessário estar sempre atualizado em relação a eles, criando novas assinaturas para nosso IDS estar sempre reconhecendo novos possíveis ataques.

O próximo capítulo abordará sobre o IDS estudado neste trabalho, o STAT, mostrando suas finalidades de ferramenta para detecção de intrusões.

CAPÍTULO 4 - STAT

Atualmente existem vários IDS, um que tem se destacado é o STAT – *State Transition Analysis Technique* – (VIGNA et al [A]. 2003). O STAT é uma metodologia para descrever penetrações de computadores por meio de cenários de ataques. Estes são representados através de uma sucessão de transições que caracteriza a evolução do estado de segurança do sistema.

O *framework* STAT provê de uma linguagem para especificação de cenários de ataques, a STATL (VIGNA et al [A]. 2003).

Uma especificação do STATL é a descrição de um cenário de ataque completo. O ataque é modelado como uma sucessão de transições de estados que traz um sistema de um estado inicial "seguro" para um estado final. Estados são usados para caracterizar instantes diferentes de um sistema durante a evolução de um ataque.

O *framework* STAT é o resultado da evolução das técnicas originais do STAT e sua aplicação para UNIX em um *framework* geral para o desenvolvimento de aplicações baseadas em eventos, tais como sistemas de detecção de intrusão (STAT, 2008).

O *framework* STAT é composto por vários elementos: As técnicas de deteção do STAT, a linguagem STATL e o núcleo STAT. Estes elementos representam os conceitos principais do *framework*. Eles serão descritos nas seções que seguem.

4.1 Técnicas de Detecção

O STAT utiliza técnicas para descrever ataques em cenários de ataques e esses cenários de ataques são utilizados para detectar comportamentos anormais do sistema.

Os cenários são compostos de estados e transições. Os estados descrevem o estado de segurança do sistema enquanto que as transições caracterizam a evolução desses estados. Por exemplo, em um cenário de ataque descrevendo uma tentativa de violar a segurança de um sistema operacional: afirmações devem estar nas propriedades dos estados, como propriedades de arquivos, identificação de usuários ou autorização de usuários.

As transições entre os estados são anotadas em ações de assinatura que representam as principais ações, e se essas forem omitidas da execução de um cenário de ataque, impedirá o ataque de ser concluído corretamente. Outro exemplo seria uma tentativa de ataque de

varredura de portas em uma rede onde uma assinatura típica deverá incluir os segmentos TCP usados para testar as portas TCP do *host*.

A Figura 2 apresenta um exemplo de diagrama de transição de estados presente no STAT onde podem ser observados os estados e as transições.

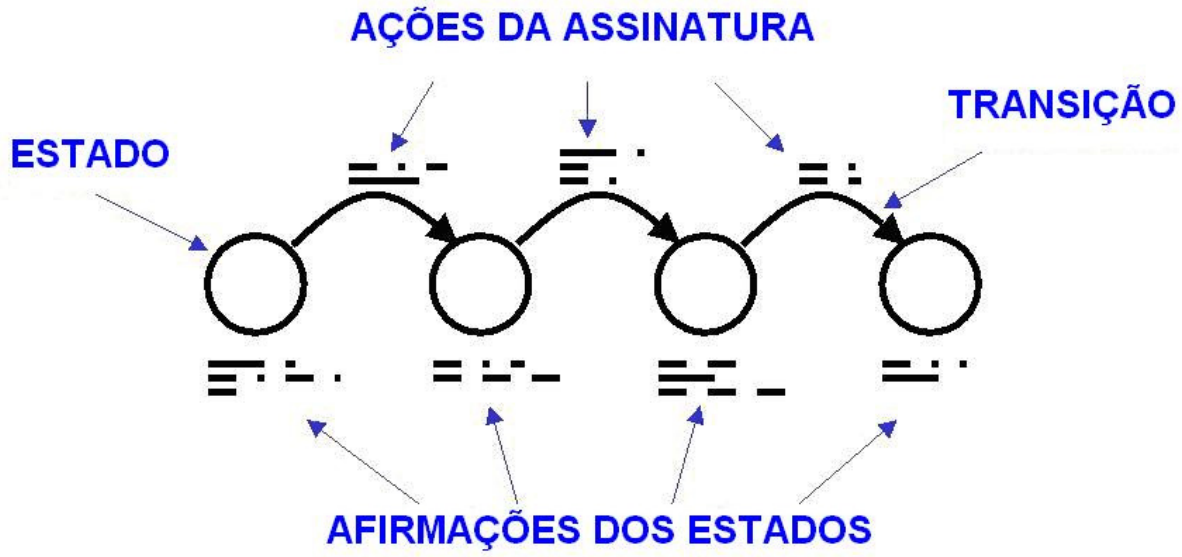

Figura 2. Diagrama das transições dos estados (STAT, 2008)

Como resultado da interpretação desses estados e transições tem-se uma linguagem, denominada STATL, que possibilita a inserção de características independentes do domínio de abordagem do STAT.

4.2 A Linguagem STATL

O *framework* STAT provê uma linguagem, chamada STATL, para a especificação de cenários de ataque. A STATL possui características independentes das técnicas do STAT e pode ser estendida para expressar as características de um determinado domínio e ambiente. O processo de extensão inclui a definição do conjunto de acontecimentos que são especificados a um determinado domínio ou ambiente a ser tratado e a definição de novas propriedades sobre esses acontecimentos (VIGNA et al [B]. 2003).

A especificação do STATL é uma descrição de um cenário completo, que é transcrito em uma representação executável que por fim é carregada na execução do IDS. A execução da especificação gera um protótipo executável que contém as estruturas de dados representando a definição do cenário e ambiente global, e um conjunto de instâncias pai/filho.

Para ter um cenário processado pelo núcleo, é necessário compilar tudo em um *plugin* de cenário (*Scenario Plugin*), que é uma biblioteca compartilhada. Além disso, cada extensão da linguagem usada pelo cenário deve ser compilada em um módulo de extensão da linguagem (*Language Extension Module*), que é uma biblioteca compartilhada também. Tanto os cenários como as extensões são traduzidas em código C++ e compiladas em bibliotecas pela ferramenta de desenvolvimento do STAT. Abaixo são apresentados os passos para se compilar um cenário STATL.

1. Crie um diretório para o *plugin* no diretório dos cenários:

```
mkdir scenarios/example
```

E copie o cenário para este diretório, em seguida, crie um arquivo chamado Makefile.am neste diretório com o seguinte conteúdo:

```
pkgname    = #scenario#
pkgver     = #version#
includes   = -I#extension-header-directory# ...
extensions = #extension-lib-relative-path# ...

statdir = @statdir@
scenariodir = $(statdir)/scenarios/$(pkgname)_$(pkgver)

EXTRA_DIST = $(pkgname).stat

scenario_LTLIBRARIES = $(pkgname).la

 INCLUDES = $(includes)

CFLAGS = -g -O2 -Wall
CPPFLAGS = -g -O2 -Wall

$(pkgname)_la_SOURCES = $(pkgname).cpp
$(pkgname)_la_LDFLAGS = -export-dynamic -module -avoid-version
$(pkgname)_la_LIBADD = -lstat
$(pkgname)_la_DEPENDENCIES = $(extensions)
```

Substitua #scenario# com o nome do seu cenário, #version# com a versão do cenário, #extension-header-directory# com o diretório onde o arquivo de cabeçalho necessário está armazenado e #extension-lib-relative-path# com o *pathname* relativo da extensão que o novo cenário depende.

Se o novo *plugin* do cenário é uma aplicação que já tem *plugins*, então o caminho mais fácil para criar um Makefile.am para o novo *plugin*, é copiá-lo de um plugin existente. Neste caso, normalmente só o nome e a versão do cenário precisam ser alterados.

2. Executar o compilador STATL para gerar um arquivo C++ a partir do cenário STATL. Por exemplo, se o cenário STATL está em um arquivo chamado example.stat, em seguida, execute:

```
statl example.stat > example.cpp
```

Assumindo que nenhum erro foi encontrado no cenário, o novo *plugin* construído no diretório agora deve conter Makefile.am, example.stat e example.cpp.

3. Antes do novo *plugin* ser compilado, o seu Makefile deve ser criado. Isto é feito por meio da inclusão do novo diretório para a lista SUBDIRS em scenarios/Makefile.am e adicionando o novo Makefile do cenário para a lista AC_OUTPUT em scenarios/configure.in. É então necessário reconfigurar, executando os seguintes comandos no diretório dos cenários:

```
automake
autoconf
./configure
```

O novo *plugin* de cenário pode agora ser construído rodando o *make* em seu diretório, ou todos os *plugins* podem ser construídos rodando o *make* no diretório *scenarios*.

4. O novo *plugin* pode ser instalado rodando o *make install* em seu diretório, ou todos os *plugins* podem ser construídos rodando o *make install* no diretório *scenarios*.

Após o *plugin* ser construído e instalado, ele pode ser carregado em um aplicativo ativado.

4.3 O Núcleo STAT

O módulo do núcleo do STAT representa o tempo de execução da linguagem STATL. O núcleo implementa os conceitos de estado, transição, instância, temporização, troca de eventos, etc. (STAT, 2008).

No tempo de execução, o núcleo é responsável pelo processo de análise de intrusão atual, obtendo eventos do ambiente alvo e comparando estes eventos com as ações e afirmações correspondentes as transições do cenário de ataque ativo. As aplicações baseadas no STAT executam análises de um ou mais fluxos de eventos (tráfego da rede, *logs* de aplicações, chamadas de sistemas, entre outros) a fim de encontrar algo suspeito. (VIGNA et al [A]. 2003)

A arquitetura de uma aplicação STAT é concentrada em torno do núcleo. O núcleo é estendido com uma série de módulos que, determinam a competência e o comportamento da aplicação. A configuração desta aplicação pode ser mudada a qualquer momento, mesmo quando estiver em tempo de execução, por meio de diretivas de controle enviadas ao núcleo. O *framework* invoca quatro elementos para a execução do núcleo (VIGNA et al [B]. 2003):

- A linguagem STATL, que é utilizada para representar cenários de ataque, por meio de estados e transições.
- Uma arquitetura *off-line* para a tradução de cenários de ataque em módulos executáveis, os chamados *Plugins* de Cenário.
- Uma arquitetura em tempo de execução para a criação dos monitores de detecção de intrusão.
- E por fim, o módulo do núcleo, que implementa as dependências semânticas de domínio do cenário em execução.

Um conjunto de módulos iniciais normalmente é definido quando se inicializa a aplicação, para produzir a sua configuração inicial. A Figura 3 apresenta uma configuração incremental de um aplicativo baseado em STAT para ilustrar melhor a função de cada módulo de sensor e descrever as dependências entre os diferentes módulos.

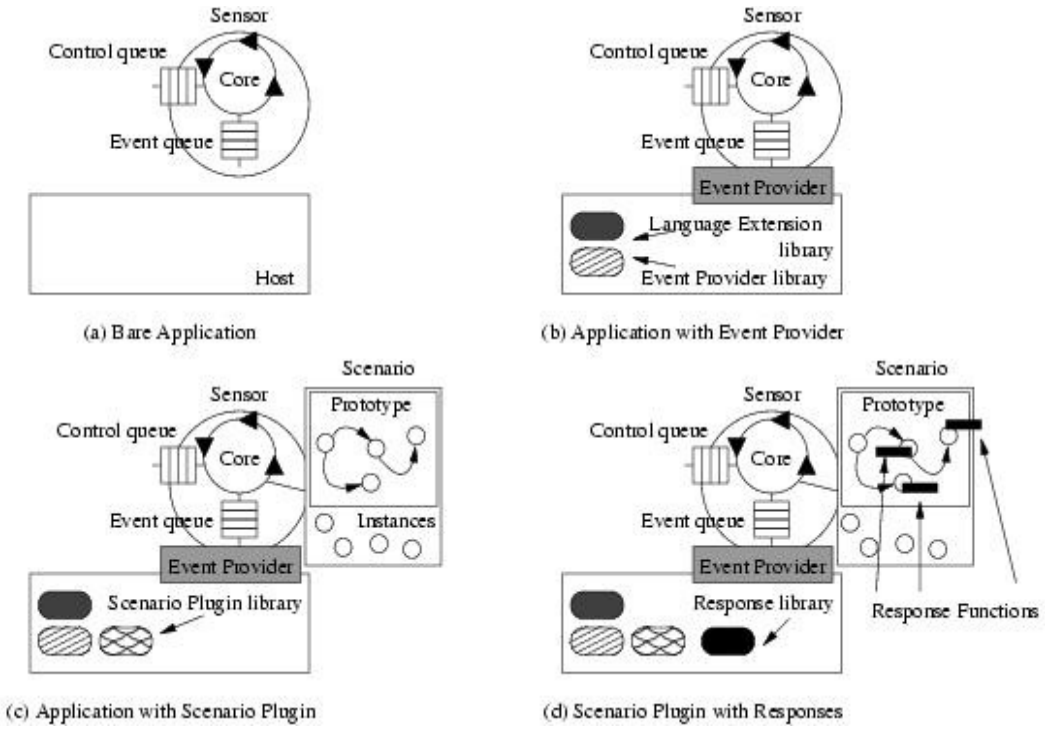

Figura 3. Evolução de uma aplicação baseada em STAT (STAT, 2008).

A Figura 3 (a) ilustra uma configuração inicial do núcleo do STAT, quando a aplicação é iniciada, não possuirá módulos, ela conterá apenas com uma instância do núcleo para controlar as mensagens que serão processadas ou espera por eventos.

Como primeiro passo faz-se necessária uma fonte de eventos. Para isso, um módulo *Event Provider* (Provedor de Evento) deve ser carregado no núcleo. Os Provedor de eventos são implementados como uma biblioteca compartilhada, sua ativação é feita por meio do envio de diretivas específicas para o controle do núcleo a fim de carregá-los e consequentemente ativá-los. O provedor reúne eventos externos a partir de fontes (exemplo, *logs* de um servidor) e transforma o resto dos eventos em objetos de eventos, encapsula-os em eventos genéricos e insere-os para o início da fila do núcleo. O núcleo, por sua vez, consome os eventos e verifica se existem cenários STAT interessados nos tipos de eventos específicos.

Os provedores podem ser adicionados dinamicamente, serem removidos e ativados mais de um evento ao mesmo tempo. Um Provedor de Eventos conta com definições de eventos contidas em um ou mais módulos de extensão da linguagem, sendo que elas devem sempre estar disponíveis às aplicações. Esta configuração está descrita na figura 3 (b).

Para começar o processamento, é necessário carregar um ou mais *Plugins* de Cenário no núcleo. Para isso, primeiro instale um *Plugin* no sensor do *host*, sob a forma de uma

biblioteca compartilhada. Ele também precisará das funções de um ou mais módulos de extensão da linguagem, que devem sempre estar disponibilizadas ao *host*. Logo após esses passos, o *Plugin* é carregado no núcleo, criando um protótipo inicial para o cenário e especificando um conjunto de parâmetros iniciais.

O protótipo do cenário contém as estruturas de dados representando as definições do cenário em termos de estados, transições, ambiente global e parâmetros de ativação. O protótipo cria uma primeira instância do cenário que estará no estado inicial correspondente ao ataque. O núcleo analisa as definições do cenário e subscreve a instância nos eventos associados com as transições que começaram a partir do estado inicial do cenário.

Neste ponto o núcleo estará pronto para processar eventos. Se um evento corresponde a uma subscrição, a afirmação correspondente à transição é avaliada. Se a afirmação for atendida, o estado será avaliado e se a afirmação dele for atendida também, a transição é terminada. Como resultado do termino da transição, a instância pode iniciar uma nova instância ou simplesmente mudar de estado. Cada instância de cenário representa um ataque em andamento. Esta situação é mostrada na Figura 3 (c), onde um *Plugin* de Cenário foi carregado e há quatro instâncias ativas no cenário.

O cenário evolui de um estado para outro e produz resultados. Um exemplo para isto é a geração de um alerta ao fim de um cenário. Outros tipos de resposta podem ser associados aos estados do cenário utilizando *Response Módules* (Módulos Resposta), que são funções que podem ser utilizadas para desempenhar qualquer tipo de resposta (por exemplo, reconfigurar um *firewall* ou o desligamento de uma conexão). Os módulos são implementados como bibliotecas compartilhadas e para ativá-los é necessário tornar a biblioteca compartilhada que contém a resposta desejada, disponível ao *host* e carregá-la ao núcleo, para então pedir a agregação da função de resposta com o estado específico na definição do cenário.

Isso permite especificar uma resposta para qualquer estado intermediário em um cenário. Cada vez que o estado especificado é atingido por qualquer uma das instâncias do cenário, a função de resposta correspondente é executada. A Figura 3 (d) ilustra um módulo de resposta e algumas funções de resposta associadas com os estados na definição do cenário.

Agora que a aplicação está totalmente configurada, Provedores de Eventos, *Plugins* de Cenário, Módulos de Extensão da Linguagem e Módulos de Resposta podem ser carregados e descarregados seguindo as necessidades da aplicação. Estas reconfigurações estão sujeitas a uma série de dependências que devem ser respeitadas a fim de conseguir carregar um componente no sensor com sucesso e tenham as entradas e saídas necessárias

disponíveis para o processamento. A gestão destas dependências é ligada à aplicação ou à infra-estrutura MetaSTAT

4.4 Ferramentas STAT

O *framework* STAT foi utilizado para desenvolver uma série de sistemas de detecção de intrusão baseados no STAT. Por exemplo, se tem:

- USTAT foi a primeira aplicação STAT baseada em intrusão de detecção no *host*. É um sistema que interpreta pistas de auditoria produzidas pela *Sun Microsystem's Basic Security Module* (BSM).
- NSTAT é uma extensão do USTAT para múltiplos *hosts*. Seu caráter distribuído permite a detecção de ataques que envolvem múltiplos *hosts* em um sistema de arquivos de rede.
- NetSTAT é um sistema de detecção de intrusão baseado em rede que analisa com o Provedor de Eventos, o tráfego de rede nos segmentos de rede procurando por algum sinal de comportamento malicioso nos pacotes que estão trafegando.
- WinSTAT é um sistema de detecção de intrusão baseado em *host* que analisa os *logs* produzidos pelo Windows NT/2000/XP, procurando por ameaças.
- WebStat é um sistema de detecção de intrusão que utilizam como entrada de dados arquivos de *logs*, analisando-os, procurando por possíveis ataques.
- AlertSTAT é um corretor de intrusão que toma como entrada de dados outros alertas de sensores e identifica os de alto nível. É usados para detectar ataques multi-passo.

4.5 A Infra-estrutura MetaSTAT

A Infra-estrutura MetaSTAT é uma infra-estrutura de comunicação entre as aplicações STAT, para controlar remotamente e coordenar as atividades de uma série de aplicações baseadas no STAT. O MetaSTAT permite reconfiguração dinâmica e gerenciamento dos sensores.

O conceito é o de que uma rede protegida é composta de uma "teia de sensores" com componentes distribuídos, integrados por uma comunicação local e pelo controle das infra-estruturas. A missão da teia de sensores é fornecer uma melhor vigilância no interior da rede protegida, utilizando-se dos *local surveillance* (locais de vigilância) contra ataques externos e

má utilização pelos usuários. As saídas dos sensores, na forma de alertas, são coletadas por uma série de "meta-sensores". Estes são responsáveis por armazenar os alertas, encaminha-los a outros sensores e meta-sensores (por exemplo, para executar a correlação de identificar cenários de ataque compostos) e para gerenciar os sensores. Cada meta-sensor fica responsável por um subconjunto de sensores e podem organizar suas atividades com outros meta-sensores. O alto nível da arquitetura da teia de sensores é mostrado na Figura 4.

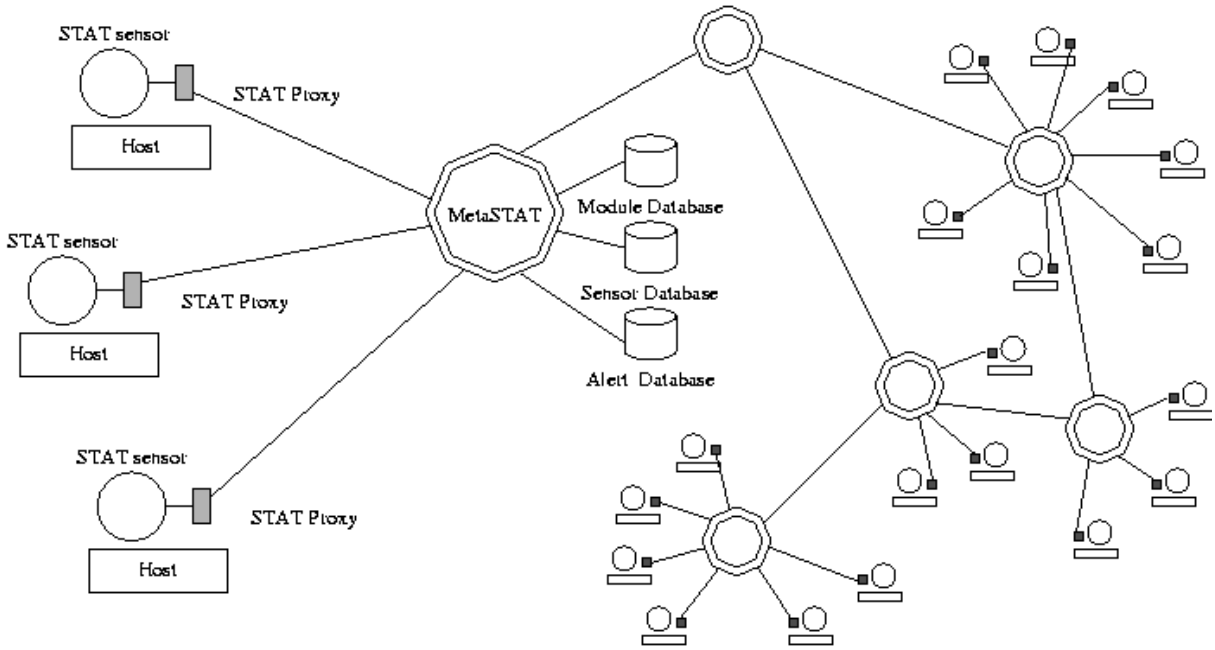

Figura 4. Arquitetura da teia de sensores (STAT, 2008)

A infra-estrutura MetaSTAT é composta por diversos módulos exercendo diferentes funcionalidades.

- A infra-estrutura CommSTAT permite troca de mensagens de alerta e controle das diretivas de forma segura entre os sensores. As mensagens CommSTAT seguem o padrão *Intrusion Detection Exchange Message Format* (IDMEF). A definição original do IDMEF inclui dois eventos *Heartbeat* e *Alert.* Este conjunto de eventos originais foi estendido para incluir controle de mensagens relacionadas ao STAT que são usadas para controlar e atualizar a configuração dos sensores;
- O controlador MetaSTAT permite ao administrador emitir controle de diretivas que modificam a configuração dos sensores inseridos, para consultar sensores remotos e obter informações sobre qual o estado em que se encontra os componentes *proxy*;

- O *proxy* do STAT atua como um intermediário entre um conjunto de sensores e um controlador MetaSTAT. Ele se conecta ao controlador utilizando CommSTAT e realiza um pré-processamento de mensagens e controle de diretivas que apóia a conexão com ferramentas de terceiros que não sejam baseadas no *framework* STAT;
- O configurador MetaSTAT possui um banco de dados dos módulos disponíveis, das dependências relativas e das configurações atualizadas do sensor. Essa capacidade de se ver implantada a teia de sensores é a base para controlar os sensores e planejar as reconfigurações da infra-estrutura de vigilância;
- O Coletor MetaSTAT é responsável por coletar e armazenar os alertas produzidos pelos sensores armazenados. Os alertas IDMEF produzidos são armazenados em um banco de dados relacional (MySQL);
- O Visualizador MetaSTAT fornece uma interface gráfica para as buscas e exibição dos alertas coletados pelo Coletor MetaSTAT.

Os componentes MetaSTAT podem ser organizados em uma estrutura hierárquica para endereçar grandes redes e controlar questões de domínio cruzado.

4.6 Considerações Finais

Este capítulo abordou o *framework* STAT que é um IDS baseado em cenários na qual possuem estados e transições para mostrar as etapas da detecção da intrusão. Foi mostrada toda sua estrutura e ferramentas para conter possíveis ataques a uma rede de computadores.

Uma vez que existe um *framework* que permite a elaboração e confecção de diferentes cenários de ataque, e que permite ainda o desenvolvimento de diferentes IDS, faz-se necessário um estudo mais elaborado de como esses cenários e essas assinaturas de ataques podem ser geradas e analisadas.

Desse modo, o próximo capítulo apresenta os testes realizados com o STAT.

CAPÍTULO 5 - DESCRIÇÕES APROFUNDADAS DO STAT E UM EXEMPLO

Este capítulo abordará sobre o STAT na prática por meio de um exemplo de cenário escrito na linguagem STATL.

5.1 Elementos Léxicos

Identificadores léxicos do STATL consistem em letras, números, *underline* (_) e devem começar com uma letra. Por exemplo, host_name ou Pedro, são identificadores. Todos os identificadores são *case-sensitive*, logo host_name e Host_Name são diferentes, eles utilizam também de notação orientada a objetos, por exemplo, em "object.atribute". Palavras-chaves não podem ser utilizadas como identificadores, scenario é uma palavra-chave, logo, não pode ser uma variável. Existem dois tipos de comentários, tudo o que for escrito entre /* e */, e qualquer texto escrito após //, até o fim da linha. Espaços em branco podem estar em qualquer lugar, menos em *tokens*, como por exemplo, palavras-chave ou identificadores.

5.2 Tipos de Dados

STATL possui vários tipos de dados: *int*, *bool*, *string*, *timeval* e *timer*. Também possui *arrays*, *vector*, *set*, *list* e *map*. Não é possível definir novos tipos de dados dentro de um cenário. Tipos de dados específicos devem ser definidos dentro das bibliotecas de extensão das aplicações, por exemplo, os cenários NetSTAT podem usar vários tipos de dados diferentes, se comparado aos cenários do USTAT, mas eles utilizam tipos de dados em comum, por exemplo, *int* ou *timeval*.

5.3 Cenário

Cenários podem utilizar nenhum ou vários tipos de bibliotecas especificas de aplicação, eventos, funções e predicados. Cada cenário possui seu nome, pode haver parâmetros, anotações, constantes, variáveis e o mais importante, contêm os estados e as transições que definem as assinaturas de ataque. Um cenário pode definir funções de suporte para ser usadas em afirmações dos estados e transições e blocos de códigos:

Scenario::=
{**use** *LibraryID*{‘,’*LibraryID*}’,’}
scenario *ScenarioID*
[*ScenarioParameters*]
‘{‘
[*FromMatter*]
{*State*|*Transition*|*NamedAction*}
‘}’
{*FunctionDefinition*}

O cenário deve possuir pelo menos uma transição e dois estados – um inicial e um final. O inicial não pode possuir nenhuma transição de entrada e o final nenhuma transição a partir dele. Os parâmetros são especificados em uma lista de identificadores:

ScenarioParameters::=
‘(‘*Parameter*{‘,’*Parameter*}’)’
Parameter::=*TypeParameterId*

Exemplo:

scenario exemplo {string host, int count)
{ … }

O cenário do exemplo possui dois parâmetros, host e count. Eles são acessados pelas instâncias dos cenários como constantes globais.

5.4 Front Matter

Os cenários podem incluir anotações e declararem constantes e variáveis.

Front Matter::=
{(*Annotation*|*ConstDecl*|*VarDecl*)}

ConstDecl::=

const *Type Constld* '=' *InitialValue*';'

VarDecl::=

[**global**] *Type VarId*['=' *InitialValue*]';'

Variáveis declaradas como globais são compartilhadas por todas as instâncias do cenário. Cada variável que não for declarada como global é instanciada privativamente em cada instância do cenário. Variáveis podem assumir valores iniciais, conforme o exemplo abaixo.

```
use netstat;
scenario example
{
    const int bufsize = 1024;
    global int count = 0;
    Host server;
    ...
}
```

Este exemplo declara uma constante inteira "bufsize" com o valor 1024 e declara uma variável global "count" com o valor inicial 0. Esta variável será compartilhada por todas as instâncias do cenário. Se qualquer instância do cenário incrementar a variável "count", sua atualização será vista por todas as outras instâncias do cenário. A declaração de variáveis do cenário também possui uma variável "server" do tipo "Host", que é um tipo de dado do NetSTAT.

5.5 Estados

Estado é um dos dois principais conceitos da linguagem STATL. Estados possuem nomes, para poderem ser referenciados em transições e na representação gráfica do cenário. Cada estado pode possuir anotações, uma afirmação e um bloco de código, sendo que são elementos opcionais:

```
State::=
    [initial]
    state StateId {Annotation}
    '{'
            [StateAssertion]
            [CodeBlock]
    '}'
```

Somente um estado pode ser designado como o estado inicial. Quando o *Plugin* de cenário é carregado no IDS, a primeira instância é criada no estado inicial.

A afirmação do estado, se presente, é testada antes de entrar no estado e ela é implicitamente *True* (Verdadeira), se não for especificada. O bloco de código do estado é executado após a afirmação da transição e a do estado ter sido avaliadas e consideradas Verdadeiras, após isso, o bloco de código da transição (se existir) é executado, conforme o exemplo abaixo.

```
scenario exemplo
{
    const int threshold = 64;
    int counter;
    ...
    initial
    state s1 { }
    ...
    state s3
    {
            counter > threshold
            { log ("counter over \ threshold limit");}
    }
    ...
}
```

Neste exemplo, o estado “s1” é denominado o estado inicial, sem uma afirmação, nem um bloco de código. O estado “s3” possui uma afirmação e um bloco de código. A afirmação especifica que o valor da variável local “counter” é maior que a variável constante “threshold”. O bloco de código chama o procedimento “log” para escrever a mensagem para o arquivo de *log* do IDS.

5.6 Transição

Transição é o segundo principal conceito da linguagem STATL. Cada transição deve possuir um nome e deve indicar o par de estados que conecta. As transições podem ter a mesma fonte e estado destino (*loops* são permitidos), como também, anotações, devem especificar um tipo de evento para satisfazer e podem ter um bloco de código.

Transition::=
transition *TransitionID*’(‘*StateId* ‘->’ *StateId* ‘)’
(**consuming**|**nonconsuming**|**unwinding**)
{*Annotation*}
‘{‘
(‘[‘ *EventSpec* ‘]’ | *ActionId*)
{*Annotation*}
[‘:’ *Assertion*]
[*CodeBlock*]
‘}’

O evento da transição é especificado diretamente ou por referência, para uma ação nomeada de assinatura. Primeiramente, a afirmação da transição é única, mas, se a ação nomeada de assinatura incluir uma afirmação e a transição também, então, a afirmação resultante é a conjunção das duas afirmações. Um exemplo de ação nomeada de assinatura está na seção 5.8, deste capítulo.

O bloco de código da transição é executado depois de avaliada a afirmação da transição e a do estado destino, e antes de executar o bloco de código do estado destino. Mais precisamente, a ordem de avaliação das afirmações e execução dos blocos de códigos é a apresentada a seguir:

1. Avaliar a afirmação da transição. Se for Verdadeira, então,
2. Avaliar a afirmação do estado. Se for Verdadeira, então,
3. Executar o bloco de código da transição, possivelmente modificando ambientes local e global;
4. Executar o bloco de código do estado, possivelmente modificando ambientes local e global.

O bloco de código da transição pode realizar qualquer computação suportada pelo STATL e a extensão IDS usada, mas é normalmente usado para copiar valores dos campos de eventos, em ambientes globais ou locais, para futura referência.

```
use ustat;
scenario example
{
    int example
{
    int userid;
    ...
    transition t2
    (s1 -> s2) nonconsuming
{
    [READ r] : r.euid != r.ruid
    { userid = r.euid; }
}
...
}
```

Neste exemplo, “t2” é uma transição não-consumente que vai do estado “s1” para o “s2”. A especificação do evento indica que a transição deve combinar eventos do tipo READ com uma condição de filtro, especificando que os campos “euid” e o “ruid” devem diferir para a transição iniciar. O bloco de código da transição copia o campo “euid” do evento “r” na variável local “userid” para futura referência.

5.7 EventSpec

"Event specs" (Especificações de eventos) são os elementos essenciais das transições. Eles especificam quais eventos (ações da assinatura) combinar e sobre quais condições.

EventSpec::=
(*BaiscEventSpec* [*SubEventSpec*]) | *TimerEvent*
BasicEventSpec::= *EventType EventId*
SubEventSpec::= '[' *EventSpec* { ',' *EventSpec* } ']'
EventType::= **ANY** | *ApplEventType* '(' *ApplEventType* {'|' *ApplEventType* } ')'

Um *event spec* é também um *basic event spec* (especificação básica de evento) opcionalmente seguido por uma *subevent spec* (especificação de sub-evento), como também é um *timer event* (temporizador de eventos) (também na seção 5.10). Um *basic event spec* identifica o "tipo" de meta-evento ANY, que combina qualquer evento ou um tipo de evento de uma aplicação especifica (exemplo, READ) ou a disjunção de tipos de eventos de uma aplicação especifica (exemplo, (UDP | TCP)), e o nome que será usado para referenciar a combinação de eventos. Um *basic event spec* que identifica um único tipo, o combina com um evento do mesmo tipo somente. Um *basic event spec* que seja a disjunção de dois ou mais tipos de eventos, combina um evento de qualquer um dos tipos na disjunção. Uma *subevent spec* identifica uma série de *event specs*. Eles combinam uma série de sub-eventos se, todas as *event specs* dentro do *subevent spec* combina um evento em uma série de sub-eventos.

Exemplo 1:

```
[(READ | WRITE) access] :
        Access.euid != access.ruid
```

Exemplo 2:

```
[IP d1 [TCP t1]] :
        D1.src == 192.168.0.1 && t1.dst == 23
```

O primeiro exemplo é um *event spec* USTAT que combina eventos READ ou WRITE (escrita e leitura). O segundo exemplo é um *event spec* (com um *subevent spec*) que combina qualquer datagrama IP contendo um segmento TCP, com endereço IP 192.168.0.1 e porta de destino 23.

O tipo ANY é eficazmente a mesma coisa que a disjunção entre todos os tipos de eventos específicos da aplicação, mas é mais fácil de especificar (e mais eficiente para implementar como um caso especial).

5.8 NamedSigAction

Uma *Named Signature Action* (ação nomeada de assinatura) possui um nome, especifica um *event spec* e pode haver anotações:

```
NamedSigAction::=
    action ActionId {Annotation}
    '{'
            ('['EventSpec']'| ActionId)
            {Annotation}
            [':' Assertion]
    '}'
```

Named signature action podem ser usadas para melhorar a claridade e manutencibilidade quando transições múltiplas têm ações similares ou idênticas, por exemplo, tendo o mesmo tipo de ação, mas afirmações ligeiramente diferentes. Neste caso, a parte comum pode ser fatorada, colocada em uma n*amed signature action* e então ser usada nas transições similares.

```
use ustat;
scenario example
{
    ...
    action a1
    {
```

```
                [WRITE r] : réuid != 0
        }
        transition t1 (s1 -> s2)
        {
                A1: r.euid != r.ruid
        }
        transition t2 (s1 -> s3)
        {
                A1: r.euid == r.ruid
        }
        ...
}
```

Neste exemplo as transições t1 e t2 usam a n*amed signature action* a1 como o *event spec* delas, mas com diferentes afirmações. Isto é equivalente a:

```
scenario exemplo
{
    ...
    transition t1 (s1 -> s2)
    {
            [WRITE r] : (r.euid != 0) && (r.euid != r.ruid)
    }
    transition t2 (s1 -> s3)
    {
            [WRITE r] : (r.euid != 0) && (r.euid == r.ruid)
    }
    …
}
```

5.9 CodeBlock

Transições e estados podem possuir *code blocks* (blocos de códigos) que são executados depois da transição correspondente e as afirmações do estado tiverem sido avaliadas e serem Verdadeiras. Um *code block* é uma seqüência de declarações.

```
CodeBlock::=
    '{'
            {statements}
    '}'
```

As declarações em um *code block* podem ser atribuições, *loops for* e *while*, *if-then-else*, chamadas de procedimentos, etc. Semanticamente, as declarações no *code block* são executadas em ordem, no contexto de ambientes globais e locais da instância do cenário na qual o bloco de código é executado.

5.10 Timers

Timers (temporizadores) são úteis para expressar ataques nas quais alguns eventos ou uma série de eventos devem (ou não) acontecer com um intervalo, seguido de outro evento ou série de eventos. *Timers* podem também ser usados para prevenir cenários "zumbis" – cenários que não possuem uma possível evolução – de desperdiçar recursos de memória.

Timers são declarados como variáveis usando o tipo *timer*. Existem *timers* locais e globais, e todos devem ser explicitamente declarados. Eles são iniciados em *code blocks* usando o procedimento *timer_start*:

```
Exemplo:

scenario exemplo:
{
    timer t1;
state s1
{
```

```
        { timer_start(t1,30); }
    }
    transition expire (s1 -> s2)
        {timer t1}
        ...
    }
```

O *code block* do estado "s1" inicia o t*imer* "t1", que irá expirar em 30 segundos. A expiração do t*imer* é tratada como um evento e estes eventos podem ser combinados usando "t*imers events*" como transição de *event specs*. Quando o t*imer* "t1" expirar, a transição "expire" irá iniciar, levando ao estado "s2".

Iniciar um t*imer* que já está "rodando", irá redefini-lo a partir do zero. Um único t*imer* pode aparecer em várias transições, cada transição ativada que tem o t*imer* "t" como seu *event spec* inicia quando o t*imer* expira.

5.11 Assertions

Assertions (afirmações) aparecem como condições de filtro nos estados e nos *event specs* (que são os elementos de combinação das transições). Afirmações STATL são construídas a partir de constantes literais, nomes de variáveis e constantes, chamadas de funções, aritmética comum e operadores relacionais. Uma afirmação STATL é avaliada em tempo de execução, no contexto de ambientes globais e locais, da instância do cenário que é avaliada.

Afirmações podem utilizar, mas não alterar, o valor de qualquer nome nos ambientes globais ou locais, além disso, as afirmações da transição podem referir aos eventos nomeados no *event spec* e para os campos destes eventos.

5.12 Annotations

Annotations (anotações) STATL provêem uma maneira padrão de entender a linguagem com características especificas de cada aplicação. As anotações são processadas por componentes especializados da arquitetura *offline* do STAT, chamada *Analyzer* (Analisador). A tarefa do *Analyser* é realizar qualquer pré-processamento especifico do IDS

no cenário, antes do cenário estar traduzido em uma representação executável. Por exemplo, o *Analyser* do NetSTAT usa as anotações do cenário STATL para determinar onde uma prova deve ser colocada em uma rede alvo e como a prova deve ser configurada para se ter certeza, que detectará o ataque descrito. O *Analyser* invoca outro componente, o *Factbase* (base de fatos), para obter informações sobre o ambiente do alvo. Por exemplo, no NetSTAT, o *Factbase* contêm informações sobre a topologia de rede e os serviços implantados.

Sintaticamente, uma anotação é um *tag* da anotação opcionalmente seguida de uma lista de expressões, todas fechadas pelos delimitadores "<*" e "*>":

```
Annotation::=
    <* Id { Expression { ',' Expression } } *>
```

Anotações podem ser unidas aos estados, transições, *event specs*, *named signature actions* e cenários. Cada IDS define toda a personalização que precisar como anotações e é esperado que o A*nalyser* associado às processem (quaisquer anotações deixadas pelo analisador serão ignoradas pelo tradutor do STATL).

Um exemplo de uma extensão de linguagem de uma aplicação específica é a anotação ENDPOINT_PORTS frequentemente usada em ações NetSTAT para especificar que duas interfaces de rede estão se comunicando. Por exemplo, o *event spec*:

```
[IP d [TCP t]]
    <* ENDPOINT_PORTS a, b *> :
    ...
```

Representa um segmento TCP/IP trocado entre interfaces de rede "a" e "b".

Anotações são unidas aos estados, transições e *named signature actions* colocando-as somente antes da abertura da entidade. Por exemplo:

```
state s1 <* annotation *> { ... }
transition t1 (s1 -> s2) consuming
    <* annotation *> { … }
```

Uma anotação no *front matter* do cenário está no escopo do cenário. Um exemplo deste tipo, é uma série de condições no cenário NetSTAT que personaliza a assinatura de ataque para um *Factbase* especifico de instalação. Eles são especificados usando a anotação NetSTAT CONSTRAINT:

```
Exemplo:
scenario exemplo ( ... )
{
    Host server;
    Service x;
    ...
    <* CONSTRAINT
            server in Network.hosts && x in server.services
    *>
    …
}
```

Neste exemplo o *tag* da anotação é o CONSTRAINT e a lista de argumentos consiste em uma expressão.

5.13 Exemplo

Este cenário é um exemplo de código do STAT, ele foi retirado de um artigo do Giovanni Vigna, que é um dos professores que estudam o *framework* STAT no *Reliable Software Group Department of Computer Science University of California* em Santa Barbara, estado da Califórnia nos Estados Unidos. Este exemplo detecta conexões TCP semi-abertas. Uma conexão TCP está em estado semi-aberto quando o servidor está esperando a mensagem de resposta (*ack*) do cliente. É neste momento que o atacante encontrará brechas para tentar uma intrusão. Este ataque é construído pelo ataque DoS, *SYN-flooding*. Seu objetivo é consumir todos os recursos dos servidores para controlar as conexões TCP, prevenindo eles de aceitarem qualquer conexão TCP. A especificação deste cenário é a apresentada a seguir.

```
use netstat;
scenario tcp(int timeout)
{
IPAddress end_vitima;
Port porta_vitima;
IPAddress end_atacante;
Port porta_atacante;
timer t0;
initial state s0 {}
transition SYN (s0 -> s1)
nonconsuming
{
[IP ip [TCP tcp]] :
(tcp.tcp_header.flags & TH_SYN) && !(tcp.tcp_header.flags & TH_ACK)
{
end_vitima =ip.header.dst;
porta_vitima =tcp.header.dst;
end_atacante =ip.header.src;
porta_atacante =tcp.header.src;
}
}
state s1
{
{ timer_start(t0, timeout); }
}
transition ACK (s1 -> s0)
unwinding
{
[IP ip [TCP tcp]] :
(ip.header.dst==end_vitima) && (tcp.header.dst==porta_vitima) &&
(ip.header.src==end_atacante) && (tcp.header.src== porta_atacante) && !(tcp.header.flags &
TH_SYN) && (tcp.header.flags & TH_ACK)
}
```

```
transition RST (s1 -> s0)
unwinding
{
[IP ip [TCP tcp]] :
(ip.header.src==end_vitima) && (tcp.header.src==porta_vitima) && (ip.header.dst== end_atacante) && (tcp.header.dst== porta_atacante) && (tcp.header.flags & TH_RST)
}
transition Timed_out (s1 -> s2)
consuming
{
timer t0;
}
state s2
{
{
HALFOPENTCP e;
e = new HALFOPENTCP(end_atacante, porta_atacante, end_vitima, porta_vitima, start);
enqueue_event(e, HALFOPENTCP, start);
}
}
}
```

Neste cenário o atacante envia o segmento TCP “SYN” para a criação de uma conexão TCP. Isto iniciará a transição “SYN”. Nesta transição o endereço IP e as portas TCP envolvidas são salvos em variáveis locais. A transição “SYN” mudará o cenário para o estado “s1” e o temporizador será iniciado usando o parâmetro timeout como variável.

No estado “s1” dois eventos podem invalidar o ataque. Primeiro, o servidor pode responder com um segmento “RST” desligando a conexão e liberando os recursos dedicados para estabelecer a conexão TCP. Segundo, o cliente envia o “ACK” final e muda a conexão para um estado estabelecido. Estes dois eventos disparam as transições RST e ACK respectivamente. As duas transições são de desvio e levam o cenário para o estado inicial, significando que a instância do cenário não corresponde mais a um ataque em progresso.

Se nenhuma das duas transições responderem, o tempo do temporizador ira esgotar e a transição “Timed_Out” levará o cenário para o estado “s2”. Neste estado um novo evento do tipo HALFOPENTCP é criado com base nas informações recolhidas nos passos anteriores do cenário. Este evento, chamado de evento sintético, é encapsulado em um evento genérico STAT e é enviado para a execução da linguagem. Ele será inserido nos eventos da rede e poderá ser processado por outros cenários. Eventos sintéticos permitem encadeamento entre cenários diferentes.

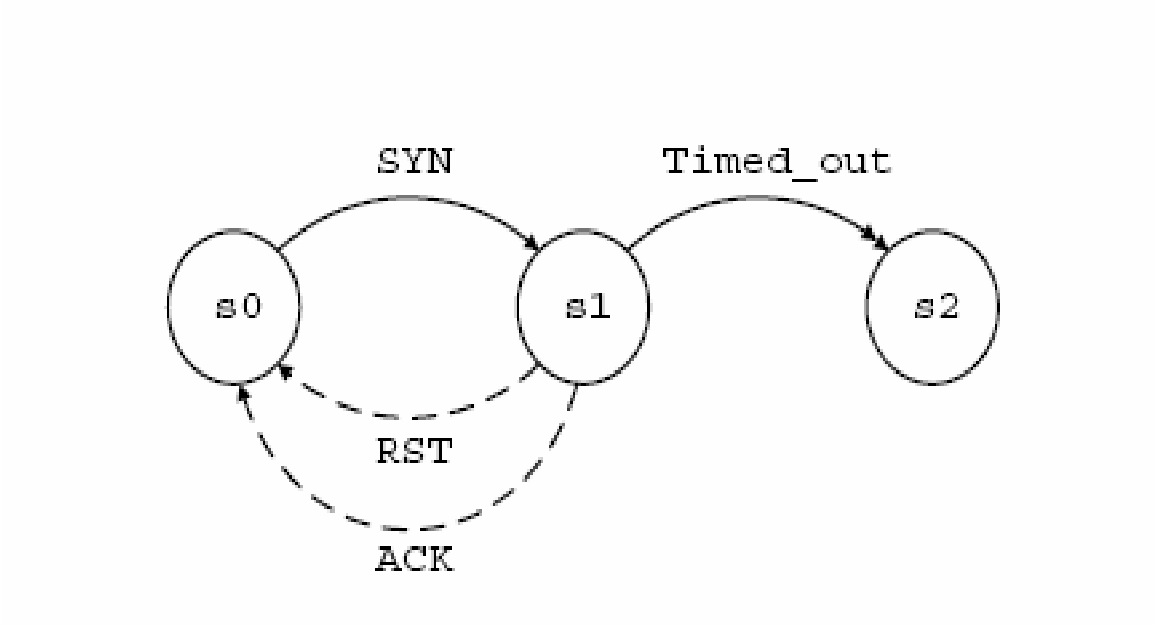

Figura 5. Representação dos estados descritos no cenário exemplificado (VIGNA et al [B], 2003).

5.14 Considerações Finais

Este capítulo abordou as especificações mais profundas do STAT, mostrando exemplos de como estar implementando cenários por meio da linguagem STATL e por fim mostrando um exemplo completo de um cenário.

CONCLUSÃO

Apesar de todo esforço para manter sua rede de computadores segura, os *hackers* estão sempre buscando e testando novas formas de encontrar vulnerabilidades contra qualquer sistema, esteja ele prevenido ou não. As técnicas utilizadas para burlar a segurança estão cada vez mais sofisticadas, permitindo que qualquer pessoa mal-intencionada as utilize.

Existem muitas maneiras para ter segurança atualmente, porém, uma das mais eficientes seria a instalação de um Sistema de Detecção de Intrusão, pois o mesmo detecta o ataque antes que algo pior aconteça.

O IDS não deve substituir nenhuma ferramenta contra ataques, mas sim ser somado a elas para garantir uma maior segurança a sua empresa. Apesar de todas as qualidades que o IDS possui, ele precisa ser muito estudado a fim de melhorá-lo para ter uma segurança sem erros, por exemplo, em falso-positivos, quando o IDS detecta como sendo uma intrusão alguma situação normal do sistema.

O objetivo deste trabalho foi realizar um estudo teórico de um IDS, especificadamente do IDS STAT, partindo desde a explicação de um IDS até os principais tipos de ataques às redes de computadores.

O interessante de estudar o STAT, e não outro IDS foi por ser uma ferramenta nova e gratuita, que facilita a sua inclusão em empresas que não querem ter um alto-custo para estarem seguros. Sua metodologia é interessante também, por ser baseada em cenários especificados por estados e transições mostrando a evolução de um sistema quando alguma coisa de errado estiver acontecendo. A linguagem STATL é teoricamente simples e é toda focada em representar cenários de ataque.

Um dos problemas do IDS é sempre estar atualizado referente aos tipos de intrusões que podem acontecer, pois após ter criado uma assinatura contendo um tipo de ataque, ela pode ficar desatualizada logo, devido ao rápido crescimento dos tipos de ataques atualmente.

Este trabalho abre portas para futuros trabalhos a fim de estarem gerando assinaturas de ataques a partir do STAT, para futuramente estarem inserido-as em um IDS. Porém a área de segurança em redes é grande, podendo existir várias outras opções para serem estudadas e exploradas, em outros trabalhos.

REFERÊNCIAS

BALASUBRAMANIYAN, J. S.; et al. **An architecture for intrusion detection using autonomous agents**. Technical Report, Department of Computer Sciences, Purdue University, COAST Laboratory TR 98/05. 1998.

B.I.S.S. - **B.I.S.S. Forums – Denial of Service Attacks**. Disponível em: <http://www.bluetack.co.uk/forums/lofiversion/index.php/t8462.html>. Acesso em: 17 mar. 2008.

CANSIAN, A.M; et al. **Um modelo adaptativo para detecção de comportamento suspeito em redes de computadores**. Brazilian Symposium on Computer Networks, São Carlos. 1997.

CARTILHA. - **Cartilha de segurança para Internet**. Disponível em: <http://cartilha.cert.br/malware/>. Acesso em: 2 mar. 2008.

CROTHERS, T. **Implementing Intrusion Detection Systems:** A Hands-on Guide for Securing the Network. Indianapolis: Wiley Publishing, 2003.

DITTRICH, D. - **David Dittrich's Home Page**. Disponível em: <http://staff.washington.edu/dittrich/ >. Acesso em: 2 mar. 2008.

ILGUN, K; et al. **State Transition Analysis**: A Rule-Based Intrusion Detection Approach. IEEE Transactions on Software Engineering, 1995.

JUNIOR, J. S. **Sistemas de Detecção de Intrusão**. Novo Hamburgo, 2003.

KUMAR, S.; SPAFFORD, E. **A software architecture to suport misuse intusion detection**. 18th National Information Security Conference, 1995.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a Internet:** uma abordagem top-down. Pearson-Addison Wesley. 3. ed. São Paulo. 2005.

MOURA, J. A. B.; et al. **Redes de computadores :** serviços, administração e segurança. São Paulo: Makron Books. 1999.

NORTHCUTT, S. **Como detectar invasão em rede.** Rio de Janeiro: Ciência Moderna, 2000.

NORTHCUTT, S.; et al. **Desvendando segurança em redes.** Rio de Janeiro: Editora Campus, 2002.

PROCTOR, P. E. **The practical intrusion detection handbook.** New Jersey: Prentice-Hall PTR, 2001.

SNORT – **The open source network intrusion detection system**. Disponível em: <http://www.snort.org>. Acesso em: 7 mar. 2008.

STAT – **STAT Home page**. Disponível em: <http://www.cs.ucsb.edu/~seclab/projects/stat/index.html>. Acesso em: 28 mar. 2008.

STANGER, J.; et al. **Rede segura Linux.** Rio de Janeiro: Alta Books, 2002.

VIGNA, G.; et al [A]. **Designing and Implementing a Family of Intrusion Detection Systems.** Reliable Software Group Department of Computer Science University of California Santa Barbara: California, 2003.

VIGNA, G.; et al [B]. **STATL: An Attack Language for State-based Intrusion Detection.** Reliable Software Group Department of Computer Science University of California Santa Barbara: California, 2003.