Uma Arquitetura para Rastreamento de Pacotes na Internet

Egon Hilgenstieler, Elias P. Duarte Jr.

[1]Departamento de Informática – Universidade Federal do Paraná
Caixa Postal 19018 Curitiba 81531-990 PR

{egon,elias}@inf.ufpr.br

Abstract. *After identifying an attack to a connected network, it's important to be able to identify the origin of the attacker. However, the Internet's architecture does not allow the IP packet source to be reliably discovered. This work presents an architecture that maintains network traffic logs using a space-efficient structured called Bloom Filter. Through appropriate extractions techniques, it's possible to identify the packet's source. Contrary to proposed techniques, the complete path the packet has taken is determined, using less storage requirements. A configurable small false positive rates is allowed.*

Resumo. *Após identificado um ataque a uma rede conectada, é importante que seja possível identificar a real origem do atacante. Entretanto, a arquitetura da Internet não permite que a origem de um pacote IP seja descoberta de maneira confiável. Este trabalho apresenta uma arquitetura que mantém logs do tráfego da rede utilizando uma estrutura de armazenamento eficiente chamada Bloom Filter. Através de técnicas de extração apropriadas, é possível identificar a origem de um pacote utilizado no ataque. Além disto, ao contrário de arquiteturas já propostas, é determinado todo o caminho percorrido pelo pacote rastreado, sendo necessário menores quantidades de armazenamento. É admitido uma pequena taxa de falsos positivos que pode ser configurada.*

1. Introdução

A segurança na Internet tem se tornado um problema crítico, com grandes quantidades de ataques reportados diariamente [1]. Existem vários mecanismos para detectar a violação de segurança de um sistema [2]. Após detectada uma invasão, um passo importante é a identificação da origem do atacante, ou do sistema que ele utiliza, para tomar as medidas defensivas necessárias para desencorajar tais ataques no futuro. Entretanto, um atacante pode facilmente ocultar a sua origem devido à estrutura do protocolo IP (*Internet Protocol*). Mesmo não havendo a intenção do atacante de ocultar seu endereço, sua real origem pode ser alterada de maneira legítima por algumas transformações que um pacote pode sofrer na rede. Logo, não pode ser considerado confiável o campo de origem de um pacote qualquer. Este artigo apresenta a proposta e implementação de uma arquitetura para rastreamento de pacotes que busca obter não apenas a sua origem mas também o caminho exato que este percorreu até o seu destino.

O objetivo de um sistema de rastreamento é, dado um pacote IP, o horário aproximado do recebimento deste pacote e o seu destino (chamado de *vítima*) construir um *caminho de ataque*. Este caminho consiste de cada roteador por onde o pacote passou no seu caminho até a vítima. Como podem haver múltiplas fontes para um mesmo pacote, é construído e retornado um *grafo de ataque* [10]. É importante definir as asserções sobre o ambiente no qual um sistema de rastreamento opera. Estas asserções servem como guia para o projeto de um sistema eficaz. São elas:

- Os pacotes IP poderem ser endereçados a mais de uma máquina, pois podem conter como seu endereço de destino um endereço de *broadcast* ou *multicast*;
- Pacotes duplicados também podem existir na rede;
- Um atacante pode ter obtido o controle de roteadores ao longo do caminho até a vítima;
- Dois pacotes enviados pelo mesmo host com o mesmo destino podem percorrer caminhos completamente diferentes;
- Os *hosts* de origem e destino são pobres em recursos, em particular a vítima de um ataque;
- Os pacotes podem ser modificados durante o processo de roteamento;
- A operação de rastreamento não é frequente;

É inevitável que um grafo de ataque possa conter um pequeno número de nodos ditos falsos positivos, que ocorrem no caso de haverem roteadores subvertidos. Ou seja, um grafo de ataque pode conter fontes que não emitiram realmente o pacote.

O restante deste artigo está organizado como segue. A seção 2 apresenta um histórico de trabalhos relacionados. A seção 3 apresenta a arquitetura proposta. Resultados experimentais são apresentados na seção 4. Conclusões seguem na seção 5.

2. Trabalhos Relacionados

Várias soluções foram propostas para determinar a rota de um *fluxo* de pacotes, em contraposição ao rastreamento de pacotes individuais. Isto pode ser feito basicamente de duas maneiras: observando o fluxo enquanto ele atravessa a rede ou tentando inferir a rota baseada no impacto do fluxo no estado da rede [4]. O primeiro trabalho proposto para inferir uma rota [4] considerou o problema de grandes fluxos de pacotes. Algumas redes candidatas eram sistematicamente inundadas com pacotes. Desta maneira é possível observar variações no fluxo de pacotes recebidos e é possível inferir a rota percorrida pelo fluxo. Para tanto é necessário ter certo conhecimento da topologia da rede e ter a capacidade de gerar grandes fluxos de dados para um enlace de uma rede arbitrária.

Outras técnicas foram propostas para observar o fluxo de pacotes. Ao fornecer informações parciais através de um subconjunto de pacotes em um fluxo, os roteadores podem permitir que a máquina final reconstrua todo o caminho percorrido pelos pacotes depois de receber uma determinada quantidade de pacotes. Dois esquemas foram propostos para comunicar as informações referentes ao caminho percorrido pelo fluxo às máquinas finais. Em [9] as informações são codificadas em campos raramente utilizados no cabeçalho do pacote IP. Já em [11] as informações de auditoria são enviadas através de mensagens ICMP (*Internet Control Message Protocol*).

Em seguida são apresentadas duas arquiteturas alternativas para o rastreamento de pacotes baseado em informações da própria rede.

2.1. A Arquitetura SPIE

Uma alternativa para o rastreamento de pacotes é deixar a própria rede com a responsabilidade de manter os estados necessários para informações de auditoria. Uma maneira de cumprir esta tarefa é simplesmente armazenando um *log* de cada pacote em vários pontos da rede para então utilizar técnicas de extração apropriadas para descobrir o caminho percorrido pelo pacote na rede. Esta técnica foi inicialmente proposta em [7]. Entretanto, a eficiência da utilização deste *log* depende do espaço disponível para armazená-lo. Tendo em vista os enlaces usados atualmente, os *logs* de pacotes iriam rapidamente crescer até atingir tamanhos impraticáveis mesmo mantendo apenas as informações referentes a um

pequeno espaço de tempo. Surge também um problema adicional: a segurança da armazenagem de pacotes completos nos roteadores.

Em [10] foi proposta uma arquitetura com o objetivo de permitir o rastreamento de pacotes IP individuais na Internet. Esta técnica utiliza a própria rede para armazenar os estados necessários para o rastreamento, também utilizando *logging* para determinar quais pacotes passaram por cada ponto da rede. No sistema descrito, chamado de SPIE (*Source Path Isolation Engine*), o problema resultante do tamanho do *log* de pacotes é resolvido utilizando-se do armazenamento de *resumos digitais* (hash) de alguns campos de um pacote IP, que o identificam unívocamente. O uso de hash também resolve o problema de privacidade que seria decorrente do armazenamento de pacotes completos em componentes da rede. As informações referentes a um pacote são mantidas por um determinado período de tempo e depois sobrescritas para guardar informações sobre pacotes mais recentes. O tamanho deste intervalo de tempo depende dos recursos dedicados ao rastreamento. O conteúdo do pacote utilizado como entrada para a função hash deve representar de maneira única um pacote IP, permitindo sua identificação em qualquer ponto do caminho percorrido por este pacote. Ao mesmo tempo, uma entrada deve ter o menor tamanho possível, por questões de desempenho.

As várias tarefas necessárias para o rastreamento de um pacote IP são separadas em três componentes no sistema SPIE. O DGA (*Data Generation Agent*) é um componente responsável por calcular o valor do hash dos pacotes que deixam o roteador e o armazenar em tabelas. Estas tabelas representam o tráfego que passou pelo roteador em um determinado intervalo de tempo e são armazenadas localmente no DGA por determinado período dependendo das limitações de recursos do roteador. Dependendo do roteador, este componente pode ser implementado por software ou como um hardware separado conectado ao roteador por alguma interface auxiliar [8]. Os dois outros componentes, o SCAR (*SPIE Collection and Reduction Agent*) e o STM (*SPIE Traceback Manager* são responsáveis por realizar os rastreamentos, sendo que o STM é o componente que gerencia todo o sistema sendo também a interface para uma entidade que pode requisitar o rastreamento de um pacote. Os três principais componentes estão ilustrados na figura 1.

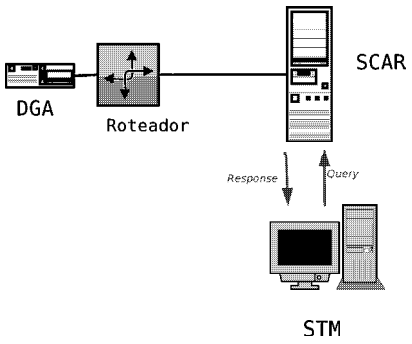

Figura 1: A arquitetura SPIE.

Antes que o processo de rastreamento possa começar, um pacote de um ataque deve ser identificado. O usuário do rastreamento pode ser, por exemplo, um sistema de detecção de intrusão que determina a ocorrência de um evento excepcional, isto é, de um ataque e fornece ao STM um pacote do ataque, a máquina alvo deste ataque (denominada *vítima*) e o horário do ataque. Dois requisitos são impostos: o primeiro é que a vítima deve ser expressa como o último *hop* e não como a máquina final em si; o segundo é que o pacote de ataque deve ser recente.

Ao receber uma requisição, o STM verifica sua autenticidade e integridade. Após uma verificação positiva, o STM imediatamente envia uma requisição a todos os SCARS em seu domínio. Estes, por sua vez, recuperaram a partir dos seus respectivos DGAs os tabelas de tráfego relevantes para análise. Cada SCAR constrói um subgrafo usando as informações da topologia sobre uma região particular da rede. Após coletar as tabelas de todos os roteadores pertencentes à sua região, o SCAR simula uma inundação do caminho reverso (*Reverse Path Flooding*) examinando as tabelas já armazenadas localmente. É importante notar que não é enviada nenhuma requisição aos roteadores pois todas as tabelas contendo as informações de tráfego foram previamente armazenadas no SCAR. Para consultar um roteador, o SCAR calcula o valor da função hash para o determinado pacote e, em seguida, consulta a tabela para verificar se existe uma entrada para este pacote. Caso positivo, é considerado que o pacote passou por este roteador. Este nodo é adicionado então ao grafo de ataque e continua para cada um de seus vizinhos, com exceção daqueles já visitados.

A figura 2 mostra um exemplo de execução do algoritmo para uma vítima V e um atacante A. O caminho percorrido pelo pacote foi A, R7, R2, R1, V.

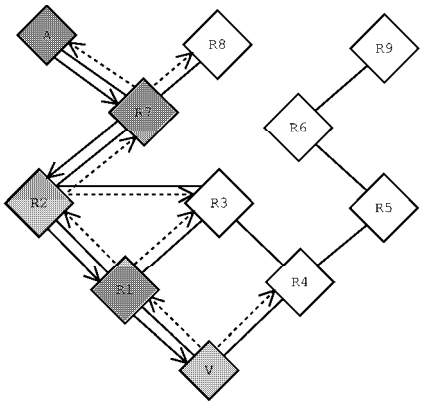

Figura 2: Exemplo de um Grafo de Ataque. As flechas contínuas representam o caminho do ataque. Flechas tracejadas representam requisições SPIE.

Se o pacote não é encontrado, a busca naquele determinado ramo da árvore de busca é terminada e a busca continua nos roteadores remanescentes. Uma lista de todos os nodos visitados é mantida para evitar ciclos na busca e garantir sua finalização. O STM se torna responsável por receber os grafos de ataques parciais de cada SCAR para poder, com base nestes, montar um grafo de ataque final.

Pacotes IP podem passar por transformações válidas durante sua travessia pela rede. O sistema SPIE deve ser capaz de reconstruir o pacote original a partir do pacote transformado. Entretanto, alguns tipos de transformações ocasionam perda de informações do pacote. Desta maneira, informações suficientes devem ser armazenadas pelo sistema SPIE para poder reconstruir os pacotes originais. Em conjunto com as tabelas armazenadas pelo DGA, o sistema SPIE mantém uma tabela de transformações para o mesmo intervalo de tempo chamada de *transform lookup table* ou TLT. Cada entrada na TLT contém três campos: O primeiro campo armazena o resultado da função hash para o referido pacote. O segundo campo especifica o tipo de transformação (três bits são suficientes para identificar os tipos de transformações descritas acima). O último campo contém informações de tamanho variado do pacote. O tamanho utilizado depende da transformação utilizada.

2.2. Uma Arquitetura Alternativa

Em [6] foi proposta uma arquitetura simples para o rastreamento de pacotes IP. Esta arquitetura também utiliza a própria rede para armazenar os estados necessários para o rastreamento. O seu funcionamento baseia-se na existência de monitores em vários pontos de observação da rede. Cada monitor mantém um registro sobre cada pacote IP observado, sem a utilização de hash. Um agente é então capaz de determinar se um determinado pacote foi ou não visto em um ponto sendo monitorado. A partir destes registros um outro componente utilizaria técnicas apropriadas de extração para descobrir o caminho que um pacote percorreu, do seu destino até a sua origem.

Muitos detalhes desta arquitetura não são tratados no trabalho apresentado. Não é especificado o algoritmo para poder reconstruir o caminho percorrido por um pacote baseado nas respostas dos monitores e nem que tipos de transformações um pacote deveria sofrer para poder ser transformado em um registro que o identificasse.

3. Uma Arquitetura para Rastreamento de Pacotes na Internet

A arquitetura proposta neste trabalho é composta basicamente por três componentes. O *Packet Monitor*, o *Packet Record Agent* (PRA) e o *Packet Tracer Application* (PTA). O *Packet Monitor* deve manter um registro ou *log* de todos os pacotes da rede de modo que seja possível verificar se um pacote qualquer pertence ao conjunto dos pacotes observados em um dado período. O PRA é o componente responsável por receber requisições para verificar se um dado pacote foi visto pelo Packet Monitor. Já o PTA é a aplicação que faz efetivamente o rastreamento de um pacote, consultando os PRA's necessários. A figura 3 ilustra todos os componentes e suas relações.

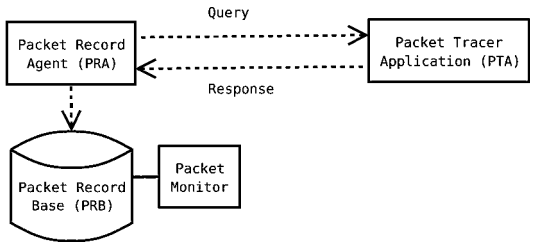

Figura 3: Componentes da arquitetura proposta.

Conforme descrito em [10], o armazenamento de pacotes inteiros em uma base de dados se torna inviável devido a grande quantidade de espaço necessário. Logo, é utilizada a mesma estratégia do componente DGA visto em [10], é calculado e armazenado apenas o resumo digital de cada pacote observado.

Apenas uma parte do pacote é usado como entrada para a função hash, de maneira que seja possível representá-lo de maneira única e mantendo o menor tamanho possível. A figura 4 mostra um pacote IP e os campos incluídos para o cálculo da função hash. O Packet Monitor calcula o hash apenas sobre a porção invariante do cabeçalho IP mais o TTL (*Time-to-Live*) e os 8 primeiros bytes do conteúdo totalizando 30 bytes. Outros campos que são frequentemente alterados são mascarados antes do resumo do pacote ser calculado. Os resultados apresentados em [10] mostram que 28 bytes (20 bytes de campos do cabeçalho e 8 bytes do conteúdo) são suficientes para identificar quase todos os pacotes

não idênticos. O componente DGA da arquitetura SPIE, por outro lado, também mascara o campo TTL do pacote antes de calcular o seu resumo digital.

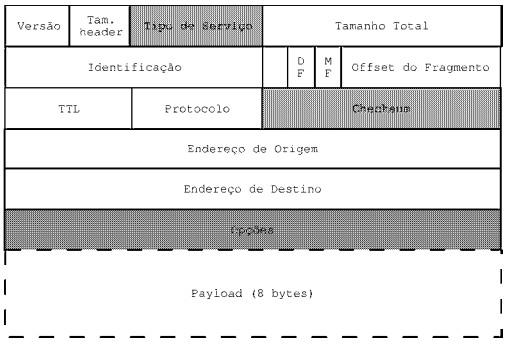

Figura 4: Os campos de um pacote IP. Campos em cinza são mascarados antes de ser aplicada a função hash.

Mesmo o armazenamento de todos os resumos digitais gerados pelo tráfego que passa pelo roteador requer quantidades muito grandes de armazenamento. Para resolver este problema, o Packet Monitor utiliza um estrutura de dados chamada *Bloom filter* [3] para armazenar os resumos de cada pacote. Um *Bloom filter* é utilizado para representar um determinado conjunto de elementos possibilitando a operação de consulta da existência de qualquer um destes elementos.

Um *Bloom filter* calcula k resumos distintos para cada pacote usando funções hash independentes. O tamanho dos resumos gerados são todos de n bits. Um vetor de 2^n bits é inicializado com zeros e as posições endereçadas pelo resultado dos resumos gerados são setados para um. Em outras palavras, se o hash(*pacote*)=y, então bloom[y]=1. A medida que outros pacotes vão chegando os bits do vetor vão sendo setados. A figura 5 ilustra a utilização de um *Bloom filter* usando k funções hash.

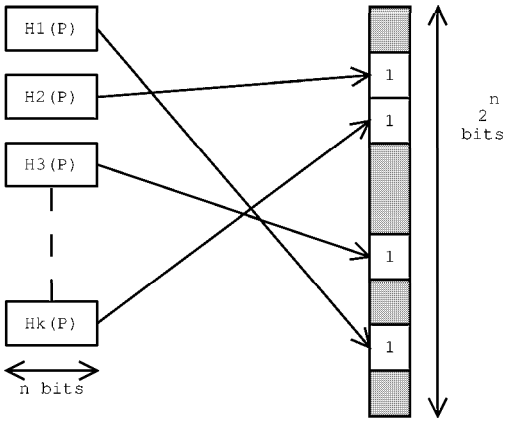

Figura 5: Utilização de um *Bloom filter*.

Um mesmo vetor é utilizado apenas por um determinado período de tempo. Depois deste período outro vetor será utilizado. O conjunto destes vetores forma uma tabela que representa o tráfego que passou pelo roteador em um determinado período de tempo.

A operação de consulta é realizada computando o resultado das k funções hash e verificando os bits correspondentes do vetor. Se os bits forem iguais a zero existe a certeza de que o pacote não foi armazenado. Entretanto, se os bits forem iguais a um, existe uma grande possibilidade de que o pacote já foi armazenado. Pode ser que outras inserções causaram estes bits a serem setados, criando assim um *falso positivo*. A taxa de ocorrência destes falsos positivos pode ser controlada [5], dependendo de k e do fator de capacidade, definido como $c = m/n$, sendo m o tamanho do vetor de bits e n o número de elementos inseridos.

Conforme descrito em [10], um sistema de rastreamento pode ser avaliado através de duas métricas. A primeira seria a precisão do grafo de ataque retornado, comparado com o grafo de ataque real. A segunda seria o período de tempo pelo qual um pacote pode ser visto. Quanto maior este período, maior o tempo em que requisições de rastreamento podem ser processadas. Este trabalho possui duas contribuições principais, a de buscar um grafo de ataque mais preciso buscando obter exatamente o caminho inverso daquele percorrido pelo pacote através do processo de roteamento e de oferecer um mecanismo para aumentar o período de tempo pelo qual os registros de pacotes podem ser mantidos, assumindo redes que possuem variações no seu nível de utilização.

Em [10], a precisão do grafo de ataque é definido em termos do número de falsos positivos que o grafo de ataque contém. Entretanto, na arquitetura SPIE podem ocorrer também o que chamamos de *falsas arestas*. Ou seja, mesmo que os nodos do grafo sejam corretos, o grafo pode conter arestas incorretas. O grafo gerado neste caso pode ser considerado incompleto e mais difícil de ser interpretado. Portanto, a precisão do grafo de ataque nesta arquitetura é definida em termos do número de falsos positivos e do número de falsas arestas do grafo de ataque gerado.

No processo de rastreamento da arquitetura SPIE, cada SCAR simula o algoritmo *Reverse Path Flooding* após coletar o vetor de bits do *Bloom Filter* de cada DGA pertencente à sua região. A partir de um DGA que tenha observado um pacote, os registros de seus vizinhos são consultados para verificar se este pacote também foi visto. Este processo se repete até que nenhum DGA vizinho restante tenha visto o pacote. Uma lista dos nós já visitados é mantida para garantir o término do algoritmo.

Entretanto, se houver uma ou mais arestas não pertencentes ao grafo de ataque mas que conectem dois nodos deste grafo (links redundantes entre nodos da rede), pode ocorrer o aparecimento de falsas arestas. Ou seja, arestas pertencentes ao grafo de ataque podem ser omitidas e podem ser incluídas arestas não pertencentes ao grafo real, conforme ilustra a figura 6. As flechas em (a) representam o caminho do ataque. Linhas contínuas representam o link entre os nodos. Em (b) é mostrado o grafo de ataque que seria retornado.

Apesar do link entre V e A não ter sido utilizado pelo pacote, ele esta contido no grafo de ataque por que ambos os nodos observaram o mesmo pacote, entretanto, foi percorrido um caminho alternativo, passando por R1. Por outro lado, a aresta que conecta R1 e A não pertence ao grafo de ataque porque R1 foi visitado após a visita a A, logo, A foi acrescentado na lista de nós já visitados, não sendo mais consultado depois de R1.

Este problema é resolvido se, ao realizar uma consulta de rastreamento, for considerado também o campo TTL do pacote IP. Sabe-se que um dado nodo X só vai preceder um outro nodo vizinho Y no grafo de ataque do pacote P, se o TTL de P observado em X for igual ao TTL de P observado em Y mais um. Desta maneira, basta fazer algumas alterações no algoritmo para busca do caminho e não mascarar o campo TTL ao inserí-lo no *log* de pacotes. Desta maneira é possível obter exatamente o caminho inverso daquele percorrido pelo pacote através do processo de roteamento.

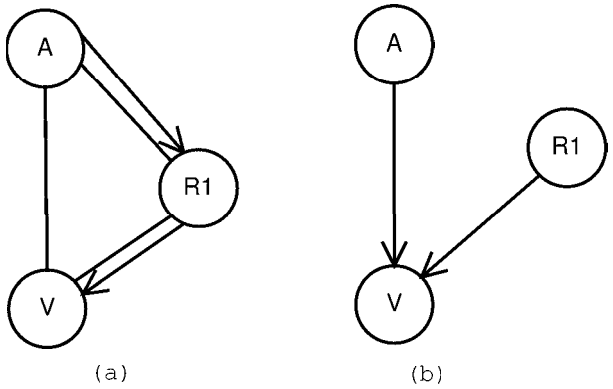

Figura 6: Exemplo de um grafo de ataque real e o obtido pelo sistema SPIE.

Para utilizar uma quantidade menor de armazenamento para o vetor de bits do *Bloom Filter* o *Packet Monitor* desta arquitetura utiliza uma outra abordagem para paginação do vetor em memória secundária. O DGA no sistema SPIE também utiliza um *Bloom Filter*, paginando o vetor de bits gerado em uma memória secundária para consultas posteriores. Entretanto, esta paginação ocorre através de um período de tempo fixo pré-determinado. O *Packet Monitor* desta arquitetura, por outro lado, recebe como parâmetro o fator de capacidade máximo, sendo paginado apenas quando atingir este valor. Desta maneira, o período de tempo em que um *Bloom Filter* permanece em memória principal é variável, podendo ser mais longo em períodos de baixa atividade da rede e mais curto em períodos de maior atividade, aproveitando ao máximo o vetor de bits utilizado.

Ao atingir o fator de capacidade máximo, o *Bloom Filter* é paginado em memória secundária junto com outras informações de controle. O conjunto de registros armazenados em memória secundária é chamado de *Packet Record Base* (PRB). O tamanho máximo que pode ser ocupado por este conjunto de registros também é determinado na configuração do *Packet Monitor*. Ao atingir o tamanho máximo, os registros mais antigos vão sendo substituídos.

O componente que verifica se um dado pacote foi visto pelo *Packet Monitor* é o *Packet Record Agent* (PRA). O PRA recebe como entrada um pacote e o horário aproximado de sua chegada. O PRA não se comunica diretamente com o *Packet Monitor*, ele apenas processa o pacote realizando as transformações necessárias, e consulta o *Packet Record Base* que é alimentado pelo *Packet Monitor*. Logo, estes dois componentes devem compartilhar a mesma região de memória. Como os registros do PRB são escritos em ordem de data, é possível utilizar uma pesquisa binária para buscar um determinado registro. O PRA também é responsável por manter informações sobre a topologia da rede. Cada PRA mantém uma lista estática de PRA's vizinhos que é definida na sua configuração. Ao receber uma requisição, o PRA responde se o pacote foi visto ou não, e caso afirmativo, informa a sua lista de vizinhos ao componente que o requisitou.

A aplicação responsável pelo rastreamento, servindo então como interface do sistema, é o *Packet Tracer Application* (PTA). Este componente precisa receber três parâmetros de entrada. O pacote a ser rastreado, o horário aproximado de sua chegada, e a vítima do pacote. A vítima deve ser expressa em termos do PRA mais próximo do host que recebeu o pacote. Com estes parâmetros, o PTA consulta o PRA recebido, que deve retornar a sua lista de PRA's vizinhos. O PTA deve consultar todos os caminhos possíveis que um pacote pode ter percorrido pois um pacote pode ter origens distintas ou um PRA pode responder com um falso positivo. O PTA realiza uma busca no grafo de ataque formado

pela resposta de cada PRA, incrementando o TTL a cada nó visitado. O procedimento acaba quando nenhum PRA restante retorna uma resposta afirmativa. Não é necessário manter uma lista dos nós já visitados pois o incremento do TTL garante o término do algoritmo.

O algoritmo para efetuar o rastreamento é mostrado na figura 7.

```
Traceback (praAnterior, praAtual, pacote, ttl)
    Se praAtual.viu_ pacote(pacote,ttl)
        imprime praAnterior, praAtual
        Para todo vizinho de praAtual
            traceback(praAtual,vizinho,ttl+1)
        Fim Para
    Fim Se
Fim
```

Figura 7: Algoritmo para rastreamento de pacotes IP.

Como é necessária uma requisição para o primeiro nodo mais próximo da vítima e como para cada nodo consultado, deve ser feita uma requisição a todos os seus vizinhos, o número de requisições necessárias para realizar o rastreamento de um dado pacote é igual ao somatório do grau de cada nodo pertencente ao grafo de ataque mais um. Logo, quanto maior o grau médio dos nodos da rede, maior será o número de requisições necessárias para completar um rastreamento. É essencial que este processo de rastreamento seja o mais rápido possível, para que as requisições sejam feitas enquanto os registros referentes ao período de tempo solicitado ainda se encontram em memória. Para esconder a latência do tempo de resposta de cada PRA, as requisições ao conjunto de vizinhos podem ser realizadas em paralelo.

4. Implementação e Resultados Experimentais

O *Packet Monitor* foi implementado em software como um *sniffer* utilizando a biblioteca PCAP (*Packet Capture*) [12]. Cada pacote lido é mascarado e inserido em um *Bloom Filter* que é armazenado em memória primária, sendo paginado em memória secundária ao atingir seu fator de capacidade máximo. A função de resumo digital utilizada foi o MD5. O seu resultado, de 128 bits, foi dividido em 8 partes de 16 bits, simulando 8 funções de resumo digital independentes. Desta maneira, cada vetor de bits do *Bloom Filter* teria 2^{16} bits ou 8 kbytes.

Para os testes foi definido um fator de capacidade máximo de 9 para o vetor de bits do *Bloom Filter*, para só então ser paginado em memória secundária. Com este valor, no máximo 7281 elementos são inseridos em um único *Bloom Filter*. Teoricamente, isto garante um índice de falsos positivos de 1.45%. Foi realizada uma simulação para confirmar o índice de falsos positivos esperado utilizando esta função. Após preencher um *Bloom Filter* até seu fator de capacidade máximo, o índice de falsos positivos foi calculado realizando testes com aproximadamente 1000000 pacotes distintos. A taxa de falsos positivos calculada foi de 1.465%, apenas ligeiramente superior ao resultado esperado. Considerando o tamanho médio de um pacote IP como sendo de 1000 bytes e assumindo total utilização do link, um *Packet Monitor* em um enlace a 100 Mbps iria processar em torno de 12500 pacotes/s. Desta maneira, dois vetores de bits poderiam representar um pouco mais de um segundo de tráfego desta rede. Seria necessário então de aproximadamente 1382400 kbytes para representar o tráfego de um dia em uma rede de 100 Mbps.

O *Packet Tracer Application* recebe como parâmetro o endereço do PRA mais próximo, que é identificado através do seu endereço IP e da porta em que ele escuta por requisições, e um arquivo binário que contém o pacote e o seu horário de recebimento. O formato deste arquivo é o mesmo utilizado pela biblioteca PCAP, que pode também ser gerado pelo utilitário *tcpdump* [12].

Foi simulada uma rede de 50 nodos executando em uma única máquina 50 instâncias do *Packet Monitor* e de PRA's executando em portas distintas. A biblioteca PCAP permite que os pacotes sejam lidos de um arquivo e não necessariamente da interface de rede. Desta maneira foi possível gerar um tráfego sintético para cada *Packet Monitor*. Um caminho de ataque foi simulado inserindo um mesmo pacote em diversos monitores da rede, variando apenas o campo TTL. Para monitores não pertencentes ao caminho de ataque, um tráfego aleatório foi utilizado.

Foram gerados 30 grafos utilizando o método de Waxman [14] para geração de grafos aleatórios que representam topologias similares às da Internet. Neste método, a probabilidade de existir uma aresta entre dois vértices varia de acordo com a distância entre tais vértices, tentando assim capturar a característica de localidade existentes nas redes reais. Dois valores, α e β são usados como parâmetros para geração do grafo. Um aumento em α aumenta o número de arestas no grafo. Um aumento em β aumenta a proporção de arestas longas sobre arestas curtas. Para cada grafo foi gerado um caminho aleatório. Foram gerados caminhos com 10, 20 e 30 nodos. Os mesmos grafos e caminhos foram utilizados para todos os experimentos.

Duas métricas foram analisadas na simulação. A primeira foi o número de requisições geradas pelo rastreamento. Esta métrica é importante pois tem influência no tempo total necessário para efetuar o rastreamento de um pacote. A segunda foi a precisão do grafo de ataque gerado. Nesta simulação, o tráfego gerado não implicava na geração de falsos positivos, apenas em falsas arestas. Foram utilizados dois algoritmos para realizar o rastreamento. O primeiro foi uma simulação do algoritmo *Reverse Path Flooding* sendo que o campo TTL do pacote era mascarado pelo *Packet Monitor*, como é feito no sistema SPIE. Neste caso, uma lista dos nodos já visitados deve ser mantida para garantir o término do algoritmo. Na segunda simulação foi utilizado o algoritmo proposto. Neste caso o campo TTL não foi mascarado. Os resultados são apresentados na tabela 1.

No pior caso da primeira simulação da arquitetura relacionada SPIE, todos os nodos recebem uma requisição. Entretanto, é possível reparar que apenas em uma simulação o grafo de ataque retornado foi idêntico ao grafo de ataque real, não contendo nenhum falsa aresta. Todos as outras simulações continham arestas a mais, que não pertenciam ao grafo de ataque original, e também arestas a menos, que pertenciam ao grafo de ataque original mas que não apareciam no grafo de ataque resultante.

O número de requisições na segunda simulação é igual ao somatório do grau dos nodos pertencentes ao grafo de ataque mais um. Logo, quanto maior o valor dos parâmetros α e quanto maior o tamanho do caminho maior será o número de requisições. É possível notar que o número de requisições é maior que o da simulação anterior. Por outro lado, o grafo de ataque retornado é exatamente o mesmo que o grafo original, não houve o aparecimento de nenhuma falsa aresta. Como o número de requisições é dependente do grau dos nodos do caminho do ataque, o tempo para processar o rastreamento pode ser reduzido bastante realizando as diversas consultas em paralelo. Desta maneira é possível esconder a latência do tempo de resposta de cada PRA.

Grafo	# nodos	Tam. caminho	Parâmetro α	Parâmetro β	# req. (sem TTL)	# req. (com TTL)	Arestas a mais	Arestas a menos	Total falsas arestas (com TTL)
0	50	10	0.808669	0.568594	50	204	4	4	0
1	50	10	0.915408	0.671377	50	211	5	5	0
2	50	10	0.231484	0.308439	29	43	0	0	0
3	50	10	0.619036	1.15522	50	197	6	6	0
4	50	10	0.175216	0.941675	34	61	3	3	0
5	50	10	0.485991	0.686354	47	129	4	3	0
6	50	10	0.324351	1.35804	45	118	4	4	0
7	50	10	0.178027	0.578125	28	54	3	3	0
8	50	10	0.855678	0.963804	50	260	5	4	0
9	50	10	0.168569	1.14614	32	56	2	2	0
10	50	20	0.106337	1.41285	41	91	4	4	0
11	50	20	0.810912	1.37801	50	568	14	15	0
12	50	20	0.350566	1.46795	50	262	8	9	0
13	50	20	0.217778	0.557753	47	113	3	3	0
14	50	20	0.987448	1.05393	50	609	14	11	0
15	50	20	0.37527	0.980862	50	269	10	10	0
16	50	20	0.916058	0.33154	50	288	11	12	0
17	50	20	0.958745	0.66182	50	489	14	14	0
18	50	20	0.150997	1.18203	45	106	5	6	0
19	50	20	0.14936	0.39847	41	112	6	6	0
20	50	30	0.273564	0.271755	50	363	16	15	0
21	50	30	0.783431	1.27042	50	813	26	24	0
22	50	30	0.433282	0.973342	50	404	15	15	0
23	50	30	0.929904	0.389298	50	495	22	25	0
24	50	30	0.78542	1.11016	50	782	25	25	0
25	50	30	0.322373	1.14768	50	321	16	15	0
26	50	30	0.972588	0.22022	50	341	23	21	0
27	50	30	0.257208	1.38542	50	279	13	16	0
28	50	30	0.723421	1.36563	50	741	22	22	0
29	50	30	0.636613	1.2483	50	640	25	23	0

Tabela 1: Resultados experimentais.

5. Conclusão e Trabalhos Futuros

Este trabalho apresentou a proposta e implementação de uma arquitetura para realizar o rastreamento de pacotes IP na Internet. São duas as principais contribuições deste trabalho. A primeira é a de retornar um caminho de ataque mais preciso. Em contraposição a trabalhos anteriores, esta arquitetura permite a determinação precisa de todo o caminho até a origem do pacote. A segunda é a proposta

de uma nova abordagem para paginação do vetor de bits utilizado no *Bloom Filter*, que é paginado apenas ao atingir um fator de capacidade máximo determinado previamente. Desta maneira, o vetor permanece em memória primária por um período de tempo maior em momentos de menor atividade da rede, reduzindo as necessidades de armazenamento mantendo a taxa de ocorrência de falsos positivos constante.

Existem algumas limitações no sistema desenvolvido que também podem ser estudados em trabalhos futuros. Devem ser considerado aspectos de segurança e autenticação nas requisições entre os componentes do sistema e permitir uma maneira de atualizar as informações sobre a topologia da rede de modo dinâmico. Estudos indicam que menos de 3% do tráfego IP passa por transformações [13]. Entretanto, um sistema de rastreamento deve ser capaz de reverter qualquer transformação realizada no pacote, pois um atacante pode tentar tirar vantagem deste fato. A reversão destas transformações não foi considerada neste trabalho. O sistema pode ser alterado para que seja possível a inserção de *plugins* que permitam o tratamento de cada tipo de transformação apropriadamente.

Referências

[1] SecurityStats.com "Security Statistics," `http://www.securitystats.com/`, acessado em 03/2004.

[2] Crothers, T., *An Overview of Intrusion Detection,Implementing Intrusion Detection Systems: A Hands-On Guide for Securing the Network*, Paperbock, 2002.

[3] Bloom, B. H. "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Communications of the ACM*, Vol.13, pp.422-426, 1970.I

[4] Burch, H. e Cheswick, B. "Tracing Anonymous Packets to Their Approximate Source," *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)*, pp.319,327, San Diego, 2000.

[5] Fan, L., Cao, P., Almeida, J. e Broder, A. Z. "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Transactions on Networking*, Vol.8, pp.281-293, 2000.

[6] Keeni, G. M. "An Architecture for Ip Packet Tracing," `http://www.ietf.org/internet-drafts/draft-glenn-ippt-arch-00.txt`, acessado em 29/10/2003.

[7] Sager G. "Security Fun with OCxmon and cflowd," `http://www.caida.org/projects/ngi/content/security/1198/mt0002.htm`, acessado em 29/10/2003.

[8] Sanchez, L. A., Milliken, W. C., *et al* "Hardware Support for a Hash-Based IP Traceback". *Second DARPA Information Survivability Conference and Exposition*, 2001.

[9] Savage, S., Wetherall, D., Karlin, A. R. e Anderson, T. "Practical Network Support for IP Traceback," *Proceedings of the ACM Special Interest Group on Data Communications 2000 (SIGCOMM'2000)*, pp.295-306, 2000.

[10] Snoeren, A. C., Partridge, C., *et al* "Hash-Based IP Traceback," *Proceedings of the ACM Special Interest Group on Data Communications 2001 (SIGCOMM'2001)*, pp 3-14, 2001.

[11] Wu, S. F., Zhang, L., Massey, D. e Mankin, A. "Intention-Driven ICMP Trace-Back," Internet Draft, IEFT, `draft-wu-itrace-intention-00.txt`, Fev. 2001.

[12] tcpdump/libpcap "TCPDUMP Public Repository," `http://www.tcpdump.org/`, acessado em 03/2004.

[13] McCreary, S. e Claffy, K. "Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange," *ITC Specialist Seminar on IP Traffic Modeling, Measurement and Management*, pp 1-11, 2000.

[14] B. M. Waxman "Routing of Multipoint Connections,"*IEEE Journal of Selected Areas in Communications*, pp 1617-1622, 1988.