Universidade do Minho
Escola de Engenharia

Mestrado em Engenharia Informática

Gestão e Segurança de Redes

SNMPv2cSec

Luís Enes Sousa
A89597

6 de setembro de 2022

Conteúdo

1 Introdução

O objetivo principal deste trabalho é implementar um agente proxy que seja capaz de estabelecer uma ligação segura entre um gestor e um agente. Os intervenientes devem comunicar usando o protocolo SNMPv2c, mesmo este não prevendo mecanismos de segurança, o que implica que terá de ser o proxy a garantir a segurança na comunicação.

Assim, o proxy deve implementar uma MIB especial de segurança que sirva de interface segura para um agente, onde o gestor coloca os seus pedidos e lê os resultados, garantindo que o gestor nunca comunica diretamente com o agente.

Esta abordagem é vantajosa, pois permite implementar a segurança em ambientes não seguros, sem ser necessária a mudança para o protocolo SNMPv3.

2 Estratégias utilizadas

Numa primeira fase, comecei por decidir a arquitetura da minha implementação. Dentro duma rede privada segura temos o agente e o proxy. O agente possui uma *SNMP RW Community*, que permite que outros hosts acedam às suas MIBs, caso tenham conhecimento da *community string*. De forma a configurar esta *SNMP Community* foi executado o sript presente na pasta 'cfg/agent'.

Neste caso apenas o proxy consegue aceder ao host, devido à configuração da rede privada. A comunicação entre o proxy e o agente é feita através do protocolo SNMPv2c. Já o proxy contém, na sua execução, um dicionário que simula a MIBSec e onde irá guardar os pedidos do gestor.

O gestor encontra-se fora da rede privada segura e comunica com o proxy, não sendo capaz de comunicar diretamente com o agente. A comunicação entre o gestor e o proxy é feita através de TCP, tanto para os pedidos do gestor como para as respostas do proxy.

Na figura seguinte podemos ver uma ilustração da arquitetura da implementação.

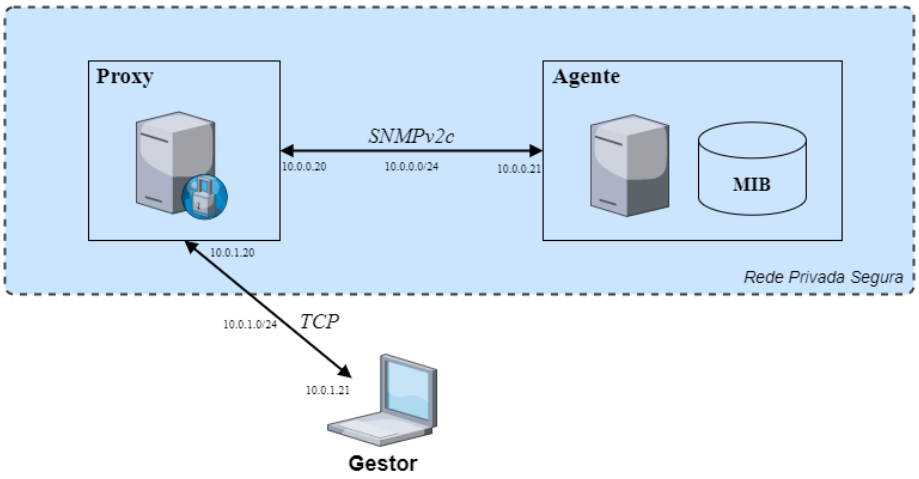

Figura 1: Arquitetura da implementação

Ao iniciar o servidor proxy, este inicializa a MIBSec e fica à escuta de conexões, no endereço 10.0.1.20. Quando um gestor se conecta, o proxy fica à espera de um pedido. Se o pedido for do tipo 'get' ou 'getnext', o proxy comunica com o agente para obter o resultado e guarda-o no dicionário que reprsenta a MIBSec. Se o pedido for do tipo 'response' então o proxy consulta os resultados guardados no dicionário e retorna o valor correspondente, ao gestor.

Quanto ao gestor, este simplesmente conecta-se ao proxy, envia os pedidos que pretende e recebe os seus resultados.

Numa fase final deste trabalho, dediquei-me a implementar mecanismos de segurança, nomeadamente autenticação dos intervenientes e confidencialidade e verificação da integridade na comunicação entre gestor e proxy.

A autenticação dos intervenientes acontece através duma troca de chaves Diffie-Hellman, que permite gerar a chave partilhada da sessão, sem comprometer as chaves privadas. De notar que esta implementação está suscetível ao ataque *Man-in-the-middle*. Depois de estabelecido este acordo de chaves, as mensagens entre o proxy e o gestor são sempre cifradas com AES-GCM, permitindo garantir a confidencialidade e autenticidade das mesmas. Os parâmetros usados na criação das chaves seguem o grupo 14 definido no RFC 3526. [1]

3 Definição da MIBSec

A definição da MIBSec encontra-se no ficheiro 'cfg/proxy/GSR2122-SEC-MIB.txt'. Esta definição começa por definir um módulo 'gsr2122' com o OID 1.3.6.1.4.101, correspondente ao caminho iso.org.dod.internet.private.101. Aqui são definidos três ramos: um onde serão definidos os objetos da MIB (1), outro onde serão definidos os eventos (2) e outro para conformidades com normas internacionais (3).

No ramo dos objetos podemos encontrar três tabelas. A tabela 'operationsTable' contém uma entrada para cada pedido do gestor e é onde será guardado o resultado desse mesmo pedido. A tabela 'agentsTable' guarda a relação entre um *alias* de um agente e o seu endereço IP e a *community string* associada. A tabela 'managersTable' funciona de maneira semelhante à tabela anterior, apenas para os gestores.

No ramo dos eventos encontra-se apenas um evento de exemplo.

No ramo das normas encontra-se outros dois ramos: um para os grupos da MIB (1) e outro para as complacências (2). O ramo dos grupos visa agrupar objetos ou eventos da MIB que estejam relacionados. Já o ramo das complacências permite indicar quais os grupos que são obrigatórios, para quem quiser implementar o módulo.

4 Acesso seguro aos agentes SNMP

Caso o gestor acedesse diretamente aos objetos das MIBs do agente através do protocolo SNMPv2c, a segurança poderia estar comprometida, pois este protocolo não prevê mecanismos de segurança, como, por exemplo, a confidencialidade, fazendo com que um atacante pudesse intercetar as comunicações e obter informações sobre o agente. Uma solução possível seria implementar uma arquitetura que usasse o protocolo SNMPv3, porém isto pode ser muito complicado em ambientes de grandes dimensões, que já estejam baseados em versões mais antigas.

Assim, com a inclusão de um agente proxy, podemos aceder aos agentes normais através do protocolo SNMPv2c, pois temos confiança na segurança da nossa rede privada. A comunicação com o ambiente não seguro (no caso deste trabalho é apenas um gestor numa rede privada, mas num ambiente realista poderia ser a Internet) é feita através de uma conexão TCP cifrada e autenticada, garantindo que o acesso à nossa rede privada segura é, também ele, seguro.

Por exemplo, quando o gestor pretende obter o valor de um dado OID do agente, é enviado um pedido pela conexão TCP ao proxy, que o valida. Depois, o proxy faz um pedido por SNMPv2c ao agente e obtém o valor, que é guardado num dicionário (que simula a MIBSec), juntamente com outras informações relativas a essa operação. O gestor receberá, então, a confirmação da conclusão da operação juntamente com o seu número identificativo, podendo enviar outro pedido ao proxy, para ler o resultado da operação, obtendo o valor associado ao OID indicado no pedido inicial.

5 Implementação

5.1 *proxy.py*

Neste ficheiro encontra-se a função *run_proxy()*, que é responsável pela execução da *Thread* principal do proxy. Esta função inicializa o dicionário que representa a MIBSec e fica à escuta por novas conexões. Quando um gestor se conecta é criada uma *Thread* responsável pela comunicação com este.

5.2 *proxy_worker.py*

Neste ficheiro encontra-se a definição da classe *ProxyWorker*, que implementa uma *Thread*. O seu método *run()* começa por fazer a troca de chaves Diffie-Hellman para obter a chave partilhada. Depois fica à espera de receber um *Packet* do gestor, que identifica o tipo de pedido e, possivelmente, mais informação adicional. Esta classe tem, ainda, mais três métodos: *process_response_request()*, *process_get_request()* e *process_get_next_request()*.

O método *process_response_request()* consulta o dicionário das operações, acedendo à entrada que corresponde ao identificador requisitado pelo gestor e retorna o valor associado a essa operação.

Já os métodos *process_get_request()* e *process_get_next_request()* executam um pedido 'get' e 'getnext', respetivamente, no agente SNMP, para cada OID especificado, e guardam o resultado no dicionário das operações.

5.3 *snmp_requester.py*

Este ficheiro disponibiliza as funções *get_request()* e *get_next_request()*, servindo de API para executar os pedidos 'get' e 'getnext', respetivamente, no agente SNMP.

5.4 *mibsec.py*

Neste ficheiro é definida a classe *MIBSec* que simula a implementação da MIBSec. Esta contém um dicionário de operações, que guarda informações como o tipo de pedido, a origem e destino e o resultado, indexadas pelo identificador da operação.

5.5 *manager.py*

Neste ficheiro encontra-se a função *run_manager()*, que é responsável pela execução do gestor. Inicialmente é estabelecida a conexão ao proxy e é feita a troca de chaves Diffie-Hellman, para obter a chave partilhada da sessão. Depois é executado um menu que permite ao gestor escolher o pedido a enviar ao proxy.

A função *process_get_results()* envia ao proxy um pedido para consultar o resultado de uma dada operação. Quando esse resultado é recebido, é exibido ao utilizador.

As funções *process_get_request()* e *process_get_next_request()* informam o proxy que deve executar um pedido 'get' ou 'getnext' no agente, respetivamente, e esperam para receber os ACKs, que indicam o identificador de cada operação.

5.6 *ctt.py*

Neste ficheiro é definida a classe *CTT*, que estabelece uma API para enviar e receber mensagens através de um socket TCP. Ainda disponibiliza uma função para serializar um objeto e outra para desserializar.

Cada instância desta classe tem associado um socket e, opcionalmente, a chave partilhada da sessão.

O método *send_msg()* começa por serializar a mensagem e calcular o tamanho resultante. Depois envia um cabeçalho, de tamanho fixo, com o tamanho da mensagem a enviar e a mensagem serializada, através do socket da instância. Caso a opção de cifrar esteja ativa, a mensagem é cifrada antes de ser serializada e é anexado ao cabeçalho o valor da sua autenticação.

O método *recv_msg()* começa por receber o cabeçalho, que indica o tamanho da mensagem a ser recebida. Depois, vai ler do socket o número de bytes indicado no cabeçalho. Por fim, desserializa os bytes recebidos. Caso a opção de cifrar esteja ativa, é verificado o valor de autenticação do cabeçalho e é decifrada a mensagem, depois de ser desserializada.

5.7 *encryption.py*

Neste ficheiro estão definidas as funções relativas à criptografia. A função *generate_HMAC()* gera o valor de autenticação de uma mensagem, usando HMAC com SHA256. A função *encrypt()* é responsável por cifrar uma mensagem, através do algoritmo AES-GCM e gerando um vetor de inicialização aleatório. A função *decrypt()* decifra uma mensagem, cifrada com a função anterior, através do vetor de inicialização, da tag do encryptor e da própria mensagem cifrada. A função *dh_key_exchange()* executa a troca de chaves Diffie-Hellman através do socket recebido.

6 Trabalho futuro

Futuramente, gostaria de implementar a conexão entre o gestor e o proxy através do protocolo SNMPv2c. Para isto seria necessário implementar a MIBSec no serviço snmpd do proxy. Só depois seria possível o gestor inserir os seus pedidos na MIBSec e consultar os seus resultados. A arquitetura da implementação mudaria um pouco, como se pode ver na figura seguinte.

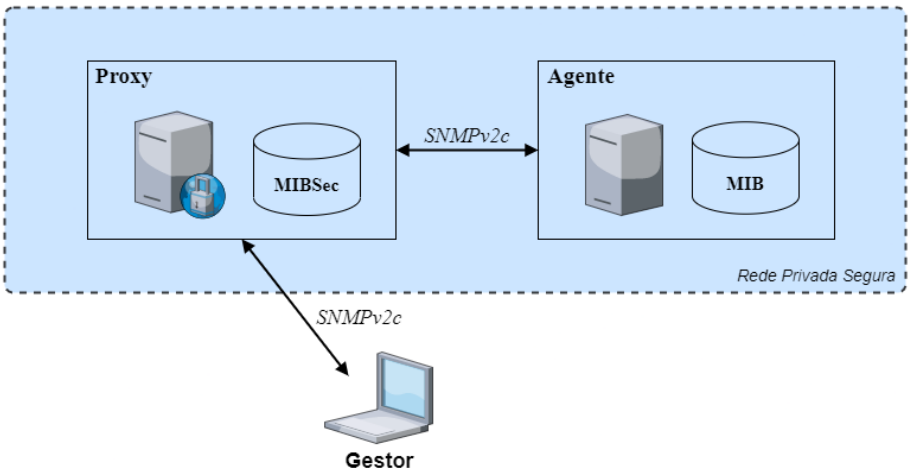

Figura 2: Futura arquitetura da implementação

A principal dificuldade que encontrei e que me impediu de prosseguir com esta arquitetura foi a criação de um *subagent* capaz de executar a MIBSec, que permitisse aceder à mesma.

No entanto, já depois de finalizar o trabalho, encontrei um módulo no GitHub que facilita este processo (`https://github.com/pief/python-netsnmpagent`). Desta forma, fui capaz de estabelecer a MIBSec no proxy e aceder através do gestor, como se pode ver na figura seguinte. Para tal, foi necessário executar o *script cfg/proxy/proxy_snmp_setup.sh* para copiar o fi-

cheiro contendo a definição da MIBSec e executar o *script cfg/proxy/run_gsr2122_agent.sh* para criar o subagente.

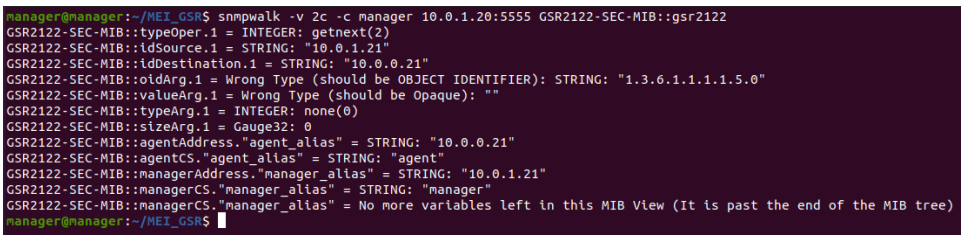

Figura 3: Snmpwalk da MIBSec através do gestor

Porém, não fui capaz de colocar a MIBSec a funcionar totalmente, pois não consegui definir os tipos 'OBJECT IDENTIFIER' e 'Opaque' através deste módulo.

7 Conclusão

Concluído este trabalho, penso que os objetivos principais foram alcançados, na medida em que consegui implementar um agente proxy capaz de estabelecer uma conexão segura entre um gestor e um agente.

Ao longo da realização deste trabalho, fui capaz de compreender, de maneira mais aprofundada, o protocolo SNMP, mais especificamente a organização em árvore das MIBs e a sua implementação. Ao ter implementado mecanismos de segurança através de conceitos criptográficos, fui, também, capaz de relembrar alguns conceitos desta área.

Referências

[1] T. Kivinen e M. Kojo. *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*. RFC 3526. RFC Editor, mai. de 2003. URL: `http://www.rfc-editor.org/rfc/rfc3526.txt`.