Covert Communications through Network Configuration Messages

Ruben Rios, Jose A. Onieva, Javier Lopez

Network, Information and Computer Security (NICS) Lab, University of Malaga, 29071, Spain

Abstract

Covert channels are a form of hidden communication that may violate the integrity of systems. Since their birth in Multi-Level Security systems in the early 70's they have evolved considerably, such that new solutions have appeared for computer networks mainly due to vague protocols specifications. In this paper we concentrate on short-range covert channels and analyze the opportunities of concealing data in various extensively used protocols today. From this analysis we observe several features that can be effectively exploited for subliminal data transmission in the Dynamic Host Configuration Protocol (*DHCP*). The result is a proof-of-concept implementation, *HIDE_DHCP*, which integrates three different covert channels each of which accommodate to different stealthiness and capacity requirements. Finally, we provide a theoretical and experimental analysis of this tool in terms of its reliability, capacity, and detectability.

Keywords: System Information Security, Network Security, Covert channels, Information Warfare, Intrusion Detection

1. Introduction

Evolution of computer networks in recent years has led to the development of new services and, simultaneously, to the emergence of new threats

Email addresses: ruben@lcc.uma.es (Ruben Rios), onieva@lcc.uma.es (Jose A. Onieva), jlm@lcc.uma.es (Javier Lopez)
[1]*Phone*: +34-952-134186; *Fax*: +34-952-134184

for interconnected systems. *Covert Channels* is a sub-discipline of Information Hiding which has been usually considered a threat to security in both centralized (e.g., Lampson (1973); Gligor (1993)) and distributed systems (e.g., Li and Ephremides (2010)). Network covert channels can be defined as a way of transmitting hidden information (i.e., unnoticed to a possible observer) using communication protocol features that are not properly defined or whose main functionality is misused. Therefore, as established, it should be possible to design covert channels at any level of the OSI stack, from the physical to the application layer.

Traditionally, covert channels have been grouped into two main categories (Brand, 1985): *storage* and *timing* channels. This classification refers to how the information to be transmitted is hidden. In the former, the sender hides data in memory areas to which the receiver has access. These memory areas might be certain header fields in network packets which are either obsolete or whose modification does not affect the proper functioning of the protocol. On the other hand, timing channels are based on modulating the behavior of the sender in order to encode information, which in practice is implemented by means of packet rate alterations.

Covert channels have not only been studied from a theoretical perspective. Several tools have been designed to hide information flows between two or more hosts. Usually, these tools take advantage of protocols widely used in most existing networks, such as TCP/IP (Rowland, 1996), HTTP (Dyatlov, 2003) or DNS (Kaminsky, 2004). Nevertheless, there are still many protocols that have not been analyzed for covert communication channels. In this work we explore various widely used protocols for short-range communications and finally develop a proof-of-concept implementation that integrates three covert channels in DHCP that may be used under different capacity and covertness requirements.

The rest of this paper is organized as follows. Section 2 provides an overview of the evolution of covert channels, from its origins to present implementations and detection mechanisms. Section 3 depicts a potential usage scenario, obtains its requirements and details the main features of the adversarial model under consideration. In Section 4 we provide an exhaustive analysis on the opportunities for information hiding in several protocols that satisfy the requirements imposed by our scenario. Next, Section 5 presents HIDE_DHCP, a new tool that integrates three forms of covert communications in DHCP resulting from the previous analysis. Moreover, Section 6 discusses on the advantages and limitations of the implemented methods, paying special attention to the tradeoff between capacity and detectability. In addition, it provides an experimental analysis on the reliability, capacity, and detectability of our tool. Finally, Section 7 concludes this work and examines future research directions.

2. Related Work

The covert channel term was first introduced by Lampson (1973) in Multi-Level Security systems to describe the ability of high security processes to signal information to other processes with lower security clearance. This concept began to draw security experts attention and was included in Brand (1985) and later in Gligor (1993) for the evaluation of systems security. Also, a chapter of McHugh (1996) was devoted to covert channels analysis and detection, where covert channels are first defined from a perspective that could be applied not only to Multi-Level Security systems but also to computer networks.

Network covert channels were originally studied in Girling (1987), where two storage channels and one timing channel were identified. This work paved the way for new studies like Wolf (1989), which analyses the IEEE 802.2 and 802.5 protocol families, and Handel and Sandford (1996) where the entire OSI reference model is discussed. The first known implementation, *Covert_TCP*, which is owed to Rowland (1996), uses three methods to hide information in the TCP and IP headers[2]. Shortly after, *LOKI2* (Daemon9, 1997), a new covert channel that uses the payload in ICMP packets, was developed. Also *PingTunnel* (Stødle, 2005) took advantage of ICMP to implement a subliminal channel. Additionally, some other well-known protocols were exploited by *FirePass* (Dyatlov, 2003) and *Ozyman* (Kaminsky, 2004), which created covert channels in HTTP and DNS respectively.

In general, the aforementioned channels benefit from misused packet headers but some other authors developed timing channels as well. The first timing channel implementation is presented in Cabuk et al. (2004), where the authors had to deal with synchronization problems due to the absence of a common precision clock. Shah et al. (2006) implement the *JitterBug*, which adds negligible delays to keystrokes in interactive network applications

[2]The vulnerabilities where found in IPv4. Later, IPv6 was found to be vulnerable to 22 forms of covert channels (Lucena et al., 2006) even before its widespread adoption.

(e.g., telnet) and these delays conceal a message that is retrieved by a remote party. Also, Luo et al. (2007) present a highly reliable timing channel, *Cloak*, that encodes a message by a unique distribution of various packets over several TCP flows. In addition, many advances have been done on the detection of timing channels based on, for example, the similarity of time intervals (Cabuk et al., 2009) and the use of entropy and conditional entropy (Gianvecchio and Wang, 2011).

Although, extensive research has been conducted in the design and implementation of covert communication channels there are still numerous protocols which are susceptible to convey hidden data either for legitimate or illegitimate purposes. In particular, we evaluate various protocols which, to the best of our knowledge, has not been exploited for such purposes yet.

3. Problem Statement

This section depicts a fictitious scenario which determines the conditions under which our covert channel should work as well as its main features. Also, it presents the capabilities of the adversary that the channel must be able to cope with.

3.1. Motivating Scenario and Attacker Model

Consider a scenario where two entities, namely *Alice* and *Bob*, want to communicate an extremely important piece of information. Alice and Bob are representatives of different countries which are involved in sensitive political or military affairs. Under such circumstances, they choose not to communicate through the Internet because of the potential threat of being observed by the government agencies from other countries.

Leveraging the fact that in the following days Bob is visiting the embassy where Alice works due to the holding of a summit of the Heads of State, Bob will try to provide her with the valuable information. The information to be transmitted to Alice is a small piece of critical information, which in case of being incorrectly received would render the covert communication useless besides posing the risk of being detected. A clear example of such a piece of information is a cryptographic key or token since the corruption of a single bit of these data result in unexpected outcomes.

During the summit it is also important that Alice and Bob do not meet alone because this might arise suspicion on the rest of attendees. Since Alice works and has privileged access to part of the network deployed in the embassy they can benefit from this issue to setup some communication channel that remains hidden to others. Moreover, the rest of attendees might monitor any transmission since they have reasons to suspect that Alice and Bob will try communicate. These are known in the literature as *passive* attackers[3] since they restrict their actions to the observation of the (timing and/or content) patterns of the traffic traversing the network. Furthermore, the attendees can not expose themselves by delaying or modifying the packets traversing the network (i.e., perform *active* attacks) because this might be considered as lack of trust in Alice and Bob and it may result in a conflict. In real-world settings, this functionality is usually implemented by means of Network Intrusion Detection Systems (NIDS), which can be qualified as global observers in the our model.

Therefore, Alice and Bob must find a way to communicate small amounts of data using an apparently innocuous channel. To that end, they must not introduce additional traffic in the network nor use unusual network protocols (i.e., setting up dedicated network channels, such as ad-hoc networks). Therefore we consider that Alice and Bob have access to the shared communication channel at any time with any of the protocols analysed in the following section and a priori raising no suspicion. Also, taking advantage of confidentiality mechanisms (i.e., encryption) is not a plausible solution since the mere observation of encrypted payloads suggests that Alice and Bob are up to something. However, encryption can be used by Alice and Bob as an extra protection to the data conveyed by the covert channel.

3.2. Covert Channel Requirements

Now we identify several properties for the hidden communication channel that should be satisfied in order to circumvent the threat of the (global and passive) attacker model under consideration:

- *Stealthiness*: Potential observers should be unable to detect the presence of a hidden communication channel. This feature is leveraged by the fact that the protocol to be chosen has not been previously used to transmit covert data.

[3]What we call attacker in this paper has been historically called Warden because of the renowned *Prisoners' Problem* (Simmons, 1983). In this respect, Alice and Bob are not considered attackers but users of a tool that allows them to communicate even in the presence of a Warden that wants to limit their rights.

- *Moderate bandwidth*: The capacity is not a critical factor since the motivation of the channel is the transmission of small amounts of information.

- *Reliability*: The data being communicated is extremely sensitive, therefore it is necessary that these are correctly received. The loss of small pieces of data may result into a great loss of information. Consider, for example, the previous scenario where the data to be transmitted is a cryptographic key.

- *Locality*: The goal is not to convey information through the Internet but to create a hidden communication channel between nodes in a local or personal area network, where there might be other entities monitoring all the communications.

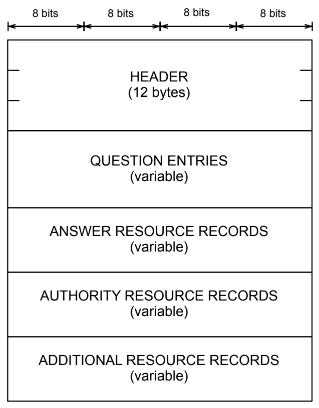
- *Unidirectionality*: The channel is not intended to allow the users to exchange information but instead transmit information that might be later used for other purposes, thus having a one-way communication channel will suffice.

Considering the aforementioned requirements, in the following we provide an analysis of various network protocols in order to find the opportunity to conceal and transmit relevant information in short-range communications. We concentrate on protocols that are incorporated in many devices and which are of widespread use in local networks such as NetBIOS, Bluetooth, and DHCP since their occurrence would not be considered as a threat. Also, those protocols should be implemented on Alice and Bob's devices fulfilling the standards and their operation cannot be altered by the proposed covert channels.

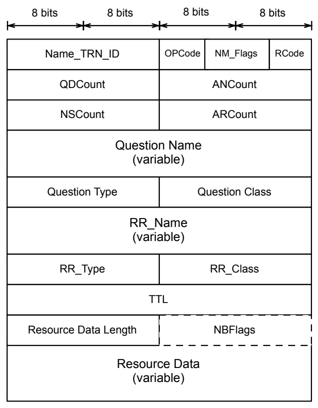
4. Analysis of Candidate Protocols

For any application of steganographic techniques the characteristics of the cover chosen for embedding data have to be investigated thoughtfully in advance. This section provides an analysis of three protocols that are commonly used in local or personal area networks, namely, NetBIOS, Bluetooth[4] and

[4]A more complete analysis of these protocols can be retrieved from `https://www.nics.uma.es/sites/default/files/TechReport-candidates.pdf`



(a) General Structure



(b) Detailed Structure

Figure 1: Name Service Packet Structure

DHCP. We provide an in-depth analysis of DHCP since our proof-of-concept implementation is based on it.

4.1. Network Basic Input/Output System

NetBIOS (*Network Basic Input/Output System*) (NetBIOS Working Group, 1987a,b) allow computers to locate resources and share them in the same local network. In order to provide these services NetBIOS defines three groups of protocols. Here we concentrate on the *Name Service* protocols mainly because in a real setting these packets are more likely than *Datagram* and *Session* packets, which are only used occasionally.

4.1.1. Name Service

The Name Service is used to register and locate resources in the network. It usually operates on the UDP port 137 and it defines 17 type of packets, whose structure is depicted in Figure 1.

Provided that there are many types packets available and that most of the fields are variable in size, it is likely that the Name Service protocols offer interesting opportunities to signal covert data. Our analysis will be focused on storage channels although several timing channels might be also exploited. For example, a node might decide whether to respond to name registration requests in order to signal 0-bit or 1-bit values to the colluding requester. Also, since any non-responded request is repeated and it is usual to observe simultaneous requests belonging to the same transaction, a covert sender could use the number of repetitions as a code. The opportunities of hiding information in this way are countless but it is necessary to deal with synchronization issues and packet loss.

The *Header* field (12 bytes) is used for identifying packets belonging to the same transaction as well as the number of entries and resource records of each type. The most relevant field here is the transaction identifier, *NAME_TRN_ID* (2 bytes), which is usually set to random value and then incremented by one for each new transaction.

Secondly, the *Question Entries* field define the name being registered, released, refreshed or queried. The *Question name* contains either a NetBIOS name or a label string pointer, which is indicated by the first byte of the field. This byte contains a particular bit pattern from which some are reserved but these are very rarely used (if ever). When the field contains a NetBIOS name and it is shorter than 16 characters, real systems complete them with space characters thus limiting the inclusion of covert data.

Finally, *Resource Records* is used to convey data about the requested resources. The general format of this field is depicted at the bottom of Figure 1b but it might present slight variations depending on the type of record (i.e., Answer, Authority, or Additional). In these fields, the most valuable field is *TTL* (4 bytes), which is used to indicate the validity period of a resource name, because slight changes in its low-order bits would pass unnoticed. Also, the *Resource Data* field contains resource information and its length is defined by *Resource Data Length*. Therefore, it might be possible to incorporate additional information after the expected contents.

4.2. Bluetooth

Bluetooth is a short-range wireless communication technology that allows the transmission of voice and data between nearby devices (Bluetooth SIG, 2012). Bluetooth defines several communication modes (i.e., profiles) but here we concentrate on Bluetooth-specific protocols at the physical and MAC layers (IEEE Computer Society, 2005), whose packet structure is depicted in Figure 2. In particular we analyse the discovery and connection between devices.

[5]The header filed is actually 54 bits long because every bit is repeated 3 times.

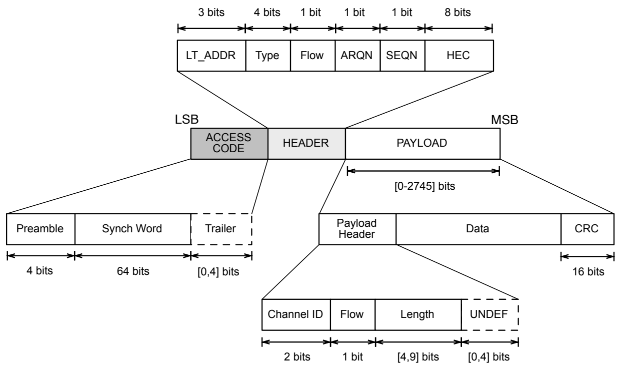


Figure 2: Standard Bluetooth Packet Format[5]

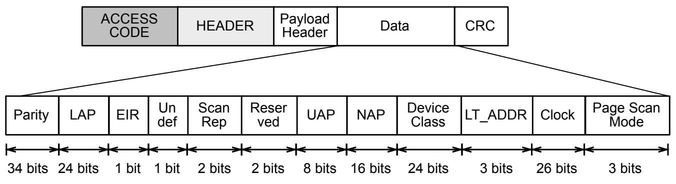


Figure 3: Detailed Format of FHS Packets

4.2.1. Device Discovery and Connection

The discovery phase allows a node to obtain the address of other devices nearby previously to its connection. In this process there are two types of messages involved. Therefore, we analyze these messages in search of fields that might be useful as covert data carriers.

- ID packet: it consists of a device access code (DAC) or inquiry access code (IAC), that is, a preamble (0101 or 1010) and a sync word (see Figure 2). The value of the preamble depends on the least significant bit (LSB) of the sync word. The sync word is derived from a 24 lower bits of the bluetooth device address (LAP). However, inquiry messages use either a general inquiry access code ($GIAC$) for the discovery of any type of device, or dedicated inquiry access codes ($DIAC$) for a specific device type, instead of the actual LAP of the device. Consequently, there is no room for hiding data in this type of packets.

- FHS packet: The structure of FHS packets is depicted in Figure 3.

  The LAP, UAP, and NAP fields constitute the physical address of the device. Modifying these values is possible but they follow a particular pattern depending on the manufacturer. Thus, a completely randomized address might alert a potential observer. Also, once the physical address is changed, it should be unaltered during the rest of the discovery and connection phase. Continuously starting unsuccessful connections would raise suspicion on other nearby devices.

The Extended Inquiry Response (*EIR*) field indicates that the device might response with an extended inquiry response packet containing more information about the services it supports. However, it is possible to send a packet with EIR set and receive no extended responses, which might be used to signal a bit of data. Moreover, the different inquiry response packets offer payloads of various sizes (up to 339 bytes). We will not analyze them further but it is worth mentioning that it is possible to signal data based on the type of packet used and the payload size.

Finally, the *Class of Device* is a 24-bit field that indicates the type of device (e.g., keyboard, PC, phone) being contacted. This is mainly used to provide the user with a graphical representation of the device. Therefore, the modification of some of these bits would not interfere with the operation of the protocol. However, from bit 23 to 13 only one bit should be set at once since they are flags that indicate the service class. Also, constinous changes on the type of device while the physical address of the device is unchanged might alert an observer.

We consider this protocol might be susceptible to new forms of covert channels, however, we will not delve further into it because we consider the analysis provided is sufficient to give a general idea of the concealment opportunities provided by Bluetooth.

4.3. Dynamic Host Configuration Protocol

DHCP (*Dynamic Host Configuration Protocol*), an auto-configuration protocol used on IP networks, can be considered as an extension of BOOTP (*Bootstrap Protocol*). It is an application-level protocol which uses UDP as its transport layer on ports 67 and 68 for the server and the client respectively. Regardless of using a datagram service, packet loss is unusual since the scope of this protocol is usually found within the same local area network. Despite this, DHCP provides some recovery mechanisms against packet loss,

Packet	**Direction**	**Description**
Discover	C $\rightarrow$ S	Broadcast message used to find servers.
Offer	C $\leftarrow$ S	Response to DHCP Discover. It contains a configuration offer.
Request	C $\rightarrow$ S	Message to confirm the acceptance of the parameters offered by the server or the renewal of a previous configuration.
Ack	C $\leftarrow$ S	Message acknowledging the parameters agreed with the client. It includes the IP address to be used.
Nak	C $\leftarrow$ S	Indicates the client the non acceptance of the configuration, either because the IP is incorrect or the lease has expired.
Decline	C $\rightarrow$ S	Message to indicate that the IP address is already in use by another client.
Release	C $\rightarrow$ S	Message to reject the given IP address, thus canceling the current lease.
Inform	C $\rightarrow$ S	Message used to request more information about the configuration; the client already has an IP address.

Table 1: DHCP Messages

such as the retransmission of packets when no response is received after a given period of time.

The DHCP protocol uses a request-response model in which the client is always in charge of starting the communication. Client-server interaction is done in transactions, where several messages are exchanged. Table 1 provides a description of the different types of messages and the direction of the communication, where C and S represent the client and the server, respectively.

Two message exchange models exist in DHCP. The first model is used the first time a client requests the network configuration parameters, or in the case the configuration lease expires. Figure 4 depicts a complete configuration process. The second model comes into play if the lease is still valid but the client is trying to renew (which usually occurs when 50% of the remaining time has been reached) that specific configuration with the server. This model can be regarded as a sub-model of the previous one (see dashed arrows in Figure 4) since it starts with a Request message aiming to renew the configuration parameters with the DHCP server. The usual exchange is as follows: Discover, Offer, Request, Ack and, optionally (dotted arrow), Release. However, the first two messages are only possible in the first model. This is important because modifying the natural communication models might alert an observer of the presence of the channel.

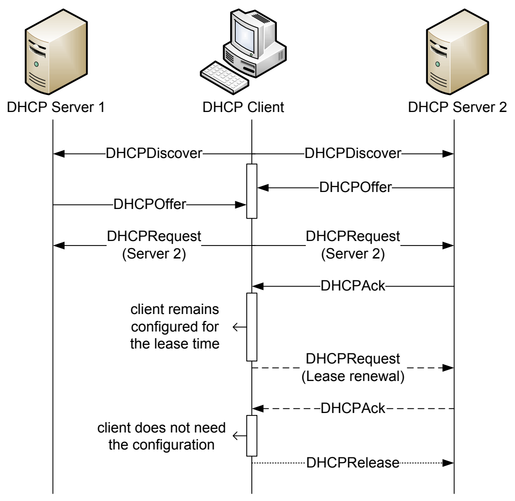


Figure 4: Message exchange models in DHCP

DHCP messages share a common header structure regardless of whether they come from the server or the client. A detailed illustration of a DHCP packet is given in Figure 5. The packet is divided into various fields which are kept for backward compatibility with the original BOOTP protocol, thus providing more chances to allocate hidden data. In the following, we analyze the fields that we consider more relevant for covert communication purposes.

First, the transaction identifier (*xid*) field has the potential to convey up to 32 bits of covert information. Interestingly, according to the protocol RFC Droms (1997) this value must be randomly created by the client. This implies that there is no agreed algorithm for the identifier generation, as with the sequence number generator in TCP, which makes the detection of the covert channel more challenging.

The field *secs* can be used in a similar way as proposed in Giffin et al. (2002) to hide information in the TCP timestamps option. The low-order bit of TCP timestamps is basically random due to host internal timings, thus it is easy to change this value to convey data without alerting a potential observer. The concealment of information without a technique such as the one proposed in Griffin's work could raise suspicions or could cause a particular server to stop working correctly upon the reception of new messages apparently delayed in time.

Also, the field *chaddr* (16 bytes) presents at least two potential ways of

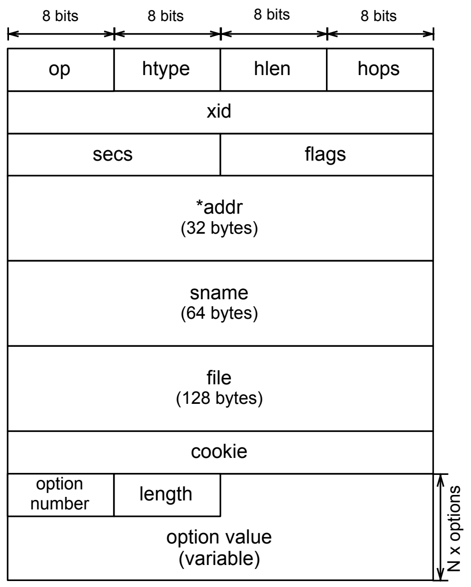


Figure 5: DHCP header format

carrying hidden data. In the first place, it could be possible to use a technique similar to the one used by Rowland (1996) in the third method developed for Covert_TCP, taking advantage of bouncing DHCP servers. In order to make this possible, the MAC address of the entity to be contacted should be included in *chaddr*. Besides, the data to be sent must be included in a header field which is not modified during the transaction, such as the *xid*. This makes the DHCP server to respond to the client specified in the *chaddr* instead of the real sender of the message, thus unknowingly helping the sender to convey the data to its destination. The only inconvenience would be having a datagram with a different MAC address to the one specified in the DHCP header. Although this is technically possible without disturbing the normal operation of the network, this could be easily identified as a spoofing attack by an intrusion detection system. The second chance for concealing data is due to the length of the *chaddr* field, which is fixed to 16 bytes while most of the times it is used for Ethernet addresses (6 bytes). Since the amount of relevant data in *chaddr* is defined by the *hlen* field, the remaining bytes can be used to convey the covert data. These remaining bytes are to be considered by ordinary servers as garbage and they do not analyze them.

The fields *sname* and *file* may carry an optional server name and a client boot file name, respectively. These fields are potentially excellent carriers of information due to their large size, 64 and 128 bytes respectively. Both fields consist of null-terminated strings ('\0'), thus our data might be included after this character without negatively impacting other clients or servers, which would consider that such fields do not contain relevant information. Although these fields are usually set to null by the operating system when they are not carrying their own data, in some situations they might contain information belonging to the Options field, which for reasons of packet capacity cannot be held within the field designed for such purpose. To indicate this situation, the RFC states that the option 52 (*Overload*) must be included. The main inconvenient in using these fields as covert information carriers is that, though not specified in the protocol definition, they are often filled with zero bytes when the *Overload* option is not active. This may alert to trained traffic analyzers.

Moreover, the *Options* field also presents interesting features which might be used to conceal information. The fact that it is a variable length field provides the ability to withhold much data, either on (1) the number of options used or (2) on the way options are ordered. Furthermore, we might also encode data by (3) placing a specific option in a particular position within the Options field. To clarify this, we provide further explanations on how each of the proposed methods could be used to encode the "ALOHA" string:

1. Number of options: for the sake of simplicity we assume that only capital letters can be sent, thus limiting the number of options to be included. To further reduce the number of options, instead of using the usual ASCII encoding we might use our own codification. For example, letter 'A' could be signaled by transmitting a message with 2 options[6], letter 'B' with 3 options, and so on. Therefore, to transmit "ALOHA" the client needs to send 5 DHCP messages, each of them with the following number of options: 2 ('A'), 13 ('L'), 16 ('O'), 9 ('H'), and 2 ('A') options respectively.
2. Options ordering: we assume that we want to encode the ASCII alphabet (8 bits) by setting options in a particular order. There are at least two ways of doing this. In the first case, we must have 8 refer-

[6]The RFC requires the occurrence of at least 2 options: *"DHCP Message Type"* and *"End"*

ence options and if a particular option appears, the bit related to that option is equal to 1 and 0 otherwise. The second way is to have a reference message to compare with: if a particular option appears in the message received in the same order as in the reference message, the bit corresponding to this option is set to 1 and 0 otherwise. For example, assume we have an alphabet comprised of 16 symbols, then we should use 4 options to encode one character. Also assume that we considered the options A, B, C and D in that particular order. If we receive a message containing C, B, A, D, then the hidden message is 0101.

3. Option type: by using the type of an option placed in a particular position, we might encode an alphabet of up to 8 bits (options types are within the range 0 to 255). Without loss of generality, let us consider the option placed in the second position as the option used to conceal the covert data. In order to send the message "ALOHA" five messages are needed and, for each of them, the second option must be option number 65 (*"NIS-Server-Addr"*), 76 (*"STDA-Server"*), 79 (*"Service Scope"*), 72 (*"WWW-Server"*) and 65; which are the ASCII equivalent codes of the string to be sent. In order to increase the capacity of this channel, instead of encoding a single symbol per packet, several different options could be used as data carriers within every packet.

One of the biggest downsides presented by the use of the above methods is that depending on the type of message there are a number of options which are either mandatory or not permitted. Therefore, the ability to deploy any of these solutions will depend on whether the introduction of unauthorized options in specific packets will influence the operation of the protocol. Another drawback, which is specific to the third proposed method, is that encoding messages with repeated characters may entail the creation of repeated options within the same packet which, although possible in some cases, may raise suspicions. However, this issue might be easily solved by sending characters only if the next character to encode has not already been included in the current message.

Finally, one might take advantage of the existence of some particular options that are either undefined or declared for private use. These options (84, 96, 102-111, 115, 126, 127, 137-149, 151-174, 178-207, 212-219, 222 and 223 are not assigned or have been erased; and 224-254 are for private use), specially those for private use, are potentially excellent information carriers since their structure has not been defined. Thus, anyone could define the *value* field in the option to be as large as necessary, up to 255 bytes.

4.4. Candidate Selection

After the analysis of the three candidate protocols for short-range communications we can conclude that DHCP is the most suitable candidate for the requirements established in Section 3.

NetBIOS opens the door to a significant number of fields that might be exploited for covert communications given the existence of 3 types of services. Also, each of the services has its own packet structure and define numerous types of messages which are intended for different situations. However, after scrutinizing these fields it was not easy to find a stealthy cover for our data. From our analysis we conclude that the most pertinent way of signaling data is by modulating the number of messages and the order of arrival at the destination given numerous control packets of this type traversing the network. However, the capacity of such channels is usually low and very susceptible to network delays. Another possibly convenient way of hiding data is introducing more entries or resource records than indicated by the Count fields, however it is unclear whether the protocol would function properly. Also, a meticulous observer would easily detect this as a threat.

In Bluetooth the main drawback is that the specifications analyzed operate at the low-layer of the protocol stack, which reduces the opportunities to hide data without perturbing the functioning of the protocol. Also, the capacity of these channels is insignificant compared to other higher level protocols. Probably, the most interesting field to conceal data is the one used to indicate the class of device but it has some limitations as discussed in Section 4.2.1.

In general, we can say that DHCP the most suitable candidate for several reasons. First, it is a renovated version of a primitive protocol and thus it is by far the most widely used protocol. Second, the structure of the packets is common for any type of message. Also, the fields are mainly large and their size is fixed regardless of their actual contents. Moreover, there are some fields which are specified to be set randomly and others reserved for private use. The main downside of this protocol is that the presence of packets from the client is determined by the lease time. All these features allows us to create various channels for different situations depending on the need for capacity or covertness.

5. Covert Channel Implementation

In this section we describe the implementation of HIDE_DHCP, which integrates 3 methods for covert communications using the *xid*, *Sname* and *File*, and *Options* fields in DHCP. The implementation is based on an already existing DHCP code, that we modified, originally developed by the Internet Systems Consortium (version 4.1.1-P1 (ISC, 2011)) and distributed in Linux operating systems.

5.1. Xid Implementation

The *xid* field is 4 bytes long and in the original ISC code it is loaded with the result of the *random()* function. This process is performed twice in the client code during the *make_discover* and *state_reboot* procedures, which are invoked every time a client requests a configuration (i.e., Discover and Request packets).

The modifications performed on the ISC code were done in such a way that the protocol specification is not altered. This results in a stealthier channel which is fully compliant with any DHCP client or server. In particular, the modified clients are able to request for network configuration parameters while transmitting hidden information to the servers and also they might behave as ordinary clients without conveying any information at all. Consequently, this choice must be notified to the modified server in order to be able to interpret the *xid* as data. In order to help the server in identifying which client is sending covert data, *start* and *end* delimiters are used (codified within the *xid* field). These delimiters are predefined but they can be modified in order to introduce some uncertainty to potential network traffic analyzers. See Section 6 for further details.

Upon the reception of a packet coming from a colluding client and containing the *start* delimiter, a new covert session starts. The received data is stored locally until the reception of the *end* delimiter. The data contained in the *xid* field is retrieved from DHCP Requests since this is a common message in both exchange models (recall Figure 4).

Furthermore, in order to enhance client-server interaction we introduced additional mechanisms to allow these entities to determine if they are communicating with the right entity, since several clients and servers might coexist in the same network and sending covert data to unmodified servers is not only useless but also might increase the detection ratio. For example, if the client does not receive a reply from the colluding server, then it will perform as an ordinary client even if he was willing to convey covert data. Nevertheless, note that this is an enhancement and that the present covert channel specification does not need client and servers identification (except for practical purposes).

5.2. Sname and File Implementation

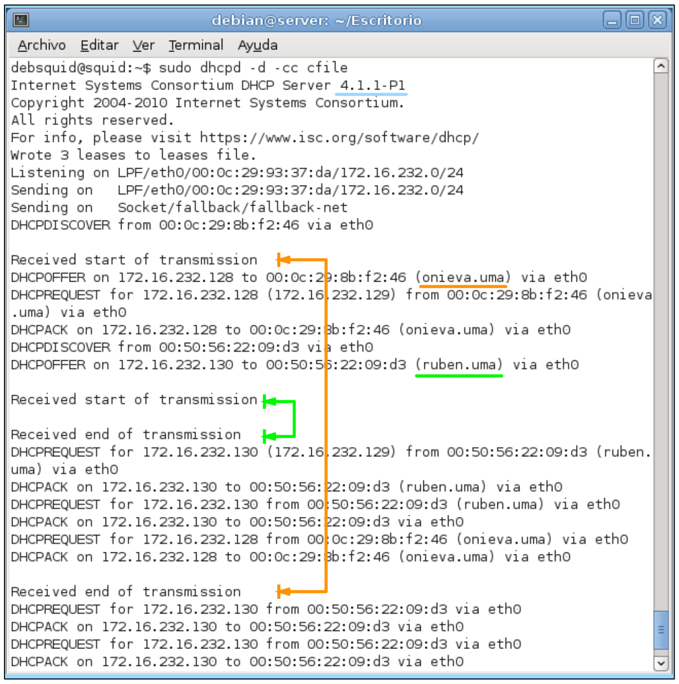
The *sname* and *file* fields present very similar features. According to the protocol specification, both fields are defined as null-terminated strings and might only carry data in the case of Discover, Inform and Request packets when coming from the client side. In case the messages are sent by the server, only Offer and Ack packets are allowed to carry data within these fields. This has been taken into consideration in order to avoid altering the normal functioning of the protocol.

The strategy in this case is to pretend to be sending empty fields by setting the first byte to the *null* character. Consequently, the implementation is able to hide a maximum of 190 bytes of data per packet, from which 63 bytes are kept within the *sname* field and 127 bytes within the *file* field. In this implementation Discover and Request packets are used as data carriers. Now we have two packets carrying data per transaction because unlike the *xid* field, they need not be constant during the whole transaction. In Figure 6 we show a snapshot of a modified server simultaneously receiving data from two colluding clients. We use the $-cc$ flag to indicate the server to accept data from clients and dump them to files.

In this implementation we also make use of delimiters because some implementations do not set to *null* these fields but instead they insert garbage. The use of delimiters prevents the modified server from storing undesired data. Also, note that the existence of implementations where the *sname* and *file* fields are filled with garbage increases the covertness of our implementation.

5.3. Options Implementation

From the various data hiding opportunities presented in Section 4.3 for the *Options* field we decided to implement a covert channel taking advantage of the options for private use. The main reason for this is that these might provide a substantial capacity compared to other choices. Another advantage of using these options is that since they are not (very often) used, the mere observation of such option in the server indicates that a client is willing to communicate.


debian@server: ~/Escritorio
Archivo Editar Ver Terminal Ayuda

```
debsquid@squid:~$ sudo dhcpd -d -cc cfile
Internet Systems Consortium DHCP Server 4.1.1-P1
Copyright 2004-2010 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Wrote 3 leases to leases file.
Listening on LPF/eth0/00:0c:29:93:37:da/172.16.232.0/24
Sending on   LPF/eth0/00:0c:29:93:37:da/172.16.232.0/24
Sending on   Socket/fallback/fallback-net
DHCPDISCOVER from 00:0c:29:8b:f2:46 via eth0

Received start of transmission
DHCPOFFER on 172.16.232.128 to 00:0c:29:8b:f2:46 (onieva.uma) via eth0
DHCPREQUEST for 172.16.232.128 (172.16.232.129) from 00:0c:29:8b:f2:46 (onieva
.uma) via eth0
DHCPACK on 172.16.232.128 to 00:0c:29:8b:f2:46 (onieva.uma) via eth0
DHCPDISCOVER from 00:50:56:22:09:d3 via eth0
DHCPOFFER on 172.16.232.130 to 00:50:56:22:09:d3 (ruben.uma) via eth0

Received start of transmission

Received end of transmission
DHCPREQUEST for 172.16.232.130 (172.16.232.129) from 00:50:56:22:09:d3 (ruben.
uma) via eth0
DHCPACK on 172.16.232.130 to 00:50:56:22:09:d3 (ruben.uma) via eth0
DHCPREQUEST for 172.16.232.130 from 00:50:56:22:09:d3 (ruben.uma) via eth0
DHCPACK on 172.16.232.130 to 00:50:56:22:09:d3 via eth0
DHCPREQUEST for 172.16.232.128 from 00:0c:29:8b:f2:46 (onieva.uma) via eth0
DHCPACK on 172.16.232.128 to 00:0c:29:8b:f2:46 (onieva.uma) via eth0

Received end of transmission
DHCPREQUEST for 172.16.232.130 from 00:50:56:22:09:d3 via eth0
DHCPACK on 172.16.232.130 to 00:50:56:22:09:d3 via eth0
DHCPREQUEST for 172.16.232.130 from 00:50:56:22:09:d3 via eth0
DHCPACK on 172.16.232.130 to 00:50:56:22:09:d3 via eth0
```

Figure 6: Simultaneous covert data reception

The modified client injects its data in the payload of an option for private use[7]. For our implementation we selected option 224 to convey our data and no delimiters are introduced. Whenever the server observes this option, it starts to store the information contained in it.

Also, since the main advantage of this implementation is the capacity to the detriment of its stealthiness, we decided to include up to 255 bytes of covert data. In case the maximum capacity of a message is reached, the Overload option is used and the remaining options are included within the *sname* and *file* fields, if they are not being used.

[7]We noticed that some options for private use are being used in an unofficial way by many DHCP servers. A commonly used option is 252 (Proxy Autodiscovery), which is used for indicating the location of the proxy server configuration parameters.

6. Implementation Analysis and Evaluation

The covert implementations presented in Section 5 have different features that allows the user to decide which method to use depending on the requirements of the scenario. There is no best covert channel under all circumstances and usually those who operate well in certain scenarios cease to be useful in others. Three properties stand out as the most important when dealing with covert channels: detectability, capacity, and reliability. In this section we review these properties from a theoretical and experimental point of view.

For the experimental evaluation we have conducted several tests in a real scenario, i.e. our laboratory. The network deployment consists of 1 server and 8 personal computers that are connected to a 24-port switch, which at the same time is connect to a router running its own unmodified DHCP server. Two additional computers are directly connected to the router, which provides Internet access to all the systems attached to it. Three out of all computers connected to the switch are used for our experiments; one of them runs our modified DHCP server and the other two run modified clients. Also, an updated version of Snort observes all the traffic from one of the client machines. The remaining computers are used as usual. For all of the experiments we generate a random key file (1024 bytes long) and send it from one (or two simultaneously) DHCP clients to the modified server using the "sname/file" implementation option[8]. Each of the experiments is executed ten times. Although ten executions are not statistically representative, packet times are very stable and packet loss is null.

6.1. Reliability

In our covert channel implementation, all the methods present optimum reliability, mainly due to the basic recovery mechanisms provided by DHCP against packet loss. In fact, due to this basic recovery mechanism used by DHCP (Droms, 1997), delays and RTT (Round Trip Time, defined as the time needed by the client to get an IP address) do not alter the covert channel reliability. This is because lease period and network delay times are in such a different order of magnitude that the DHCP client will ultimately succeeds in communicating with the server (and vice versa). It is therefore easy in

[8]The results obtained from this evaluation can be extended to the other two covert channel implementations.

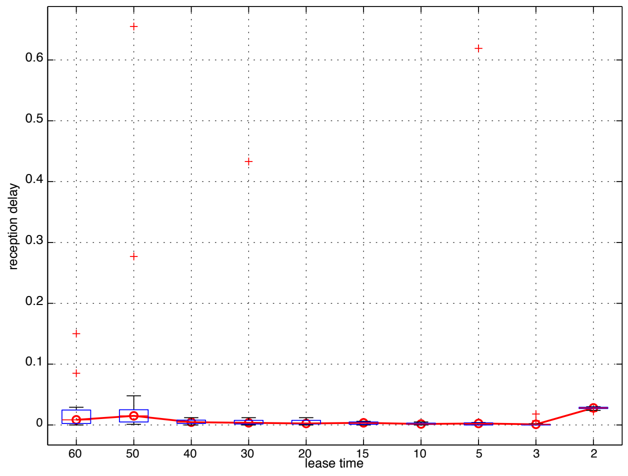


Figure 7: Packet reception delay

our implementation to identify whether the received packet is repeated or belongs to a renewal process[9].

We have defined a series of tests where we reduce the lease time from 60 seconds down to 0 seconds in order to evaluate the reliability of the DHCP server under heavy load (i.e., *stress*) situations. In this context we are interested in the potential occurrence of packet losses, out of sequence packet arrival or data repetitions. In Figure 7 we appreciate packets transmission delays observing that there no collisions or retransmissions with a traffic analyzer. In general, the median delays are very close to zero but there are some isolated cases in which the delay is of the order of several tenths of a second. Nonetheless, given the granularity of the lease period compared to these minor delays, this causes no side-effects in the reception of covert data.

6.2. Capacity

Similarly to Wu et al. (2012), we define the capacity of a channel as its maximum error-free information rate in bytes per hour (bytes/hour). Our implementations features different capacities (the capacity is usually at odds

[9]Unless the lease period is short enough to harden this task. Nevertheless, note that, although establishing a lease period shorter than in real scenarios would significantly increase the capacity, it would also increase detectability to a limit which will make the requirements fulfilment not possible.

with the ability to pass unnoticed and our solutions also present this trade-off) all of which are affected by the leasing period and this depends on the network infrastructure deployed. Highly stable networks with a large IP addresses pool can set the default lease period to eight days, whereas highly dynamic and limited networks may operate in better conditions with a period of one hour (let us assume this default case for capacity analysis). The lease period determines covert channel capacity as follows[10]:

$$Capacity \approx 2 * Maximum\ size\ per\ session\ /\ leasing\ period \qquad (1)$$

The maximum capacity parameter depends on the implementation method selected and on whether the protocol state belongs to the first or second model of the DHCP specification (see Figure 4). The *xid* method provides a very limited capacity (8 bytes/hour). Thus, this covert channel is recommended for the transmission of small amounts of highly sensitive data, for example an elliptic curve cryptography key.

The *sname* and *file* implementation provides a better capacity. It theoretically allows for the transmission of 760 bytes/hour (recall we hide data in Discover and Request packets) in the first period and 380 bytes/hour in the following ones.

On the other hand, the *options* implementation provides the higher capacity (1020 bytes/hour in the first period and 510 bytes/hour in the following ones) and is recommended for loosely supervised networks because the size of the resulting DHCP messages might attract the attention of casual observers. This method would also allow the creation of a bidirectional covert channel because option 224 is not used for any other purposes.

Table 2 represents DHCP request packets reception as detected by a traffic analyzer[11]. As expected, in the first transmission 377 bytes (the total amount of data sent is 380 bytes but our delimiter is 3 bytes long) are received while 190 bytes are then sent by the client in each renewal to complete the 1024 bytes original file. Interestingly, when the lease time is set to 0, the client behaves as if the lease offered by the server is already expired and uses the complete message exchange model (using both the discover and request messages in each session).

[10]Recall that the renewal of the session takes place approximately in the middle of the lease time period.

[11]`http://www.wireshark.org/` using the traffic filter bootp.dhcp

	Lease Time								
	60	50	40	30	20	10	5	3	2
	82,808	771,267	1377,492	1806,423	2390,329	2958,233	3247,808	5394,831	5812,842
	106,723	796,612	1397,493	1821,423	2400,332	2965,235	3250,189	5395,831	5812,870
Exp. 1	132,721	820,606	1417,504	1836,430	2410,330	2972,234	3254,189	5396,831	5812,899
	162,720	845,603	1437,500	1852,428	2420,328	2978,233	3258,185	5398,830	5812,926
	188,712	866,595	1454,494	1866,424	2432,328	2984,232	3262,185	5399,831	5812,953
	340,808	896,639	1474,477	2141,808	2455,317	3024,223	3314,176	5494,808	5916,808
	371,658	922,591	1493,477	2154,375	2467,325	3029,225	3317,177	5495,815	5916,838
Exp. 2	395,655	947,565	1511,472	2166,372	2477,319	3034,222	3320,173	5497,814	5916,862
	427,650	971,561	1530,484	2181,361	2486,317	3039,222	3323,173	5499,814	5916,890
	452,650	996,578	1550,482	2193,368	2496,317	3045,218	3327,173	5500,815	5916,917
	477,641	1021,552	1584,462	2216,356	2517,306	3092,212	3378,163	5561,805	6017,808
	504,664	1047,551	1605,459	2232,368	2528,315	3099,213	3382,166	5563,803	6017,838
Exp. 3	528,644	1071,564	1624,469	2244,360	2539,303	3105,208	3386,163	5565,803	6017,860
	552,650	1093,540	1644,466	2258,357	2550,308	3111,209	3390,163	5567,803	6017,889
	583,624	1119,559	1660,460	2272,357	2561,299	3117,210	3394,162	5569,802	6017,917
	622,620	1243,808	1684,443	2298,343	2610,289	3144,203	3447,153	5641,808	6182,808
	650,640	1266,531	1701,443	2313,342	2620,296	3150,205	3451,155	5643,790	6182,838
Exp. 4	676,631	1291,529	1722,454	2328,346	2630,294	3156,205	3454,151	5645,790	6182,862
	704,629	1313,505	1742,450	2340,344	2641,293	3163,202	3458,154	5646,790	6182,890
	735,600	1334,518	1760,445	2354,344	2653,292	3169,199	3462,151	5647,789	6182,919

Table 2: Sname/file packet arrival times

In Figure 8 we show the relationship between the lease time period proposed by the server and the actual renewal times. As aforementioned, the renewal of the current lease occurs approximately in the middle of the lease period. However, the client decides to deliberately delay or anticipate the transmission. The reason for this is to avoid overloading the server when there are many clients in the network.

Finally, a combination of all implemented techniques can be used for covert channel construction. That means 1788 bytes/hour in the first period and 898 bytes/hour in the following ones. Nevertheless, this would probably increase the detection probability.

6.3. Detectability

In terms of detectability, the three proposed methods have the advantage of being the first to use DHCP as a data carrier and moreover they strictly follow the protocol specification. Still, a smart observer might be alerted by some unusual patterns.

In our tests, none of the implementations have been detected by a regular and updated installation of a well known IDS such as Snort (Snort, 2012). Although active wardens could be deployed in order to eliminate covert channels in the local network by introducing noise (up to the MRF, Minimal Requisite Fidelity); this technique needs a previous protocol specification analysis, and to the best of our knowledge, no active wardens have been specified for the DHCP protocol yet. Nevertheless, in both cases, if DHCP-specific IDS rules and an active warden are designed for covert channel detection, discovery

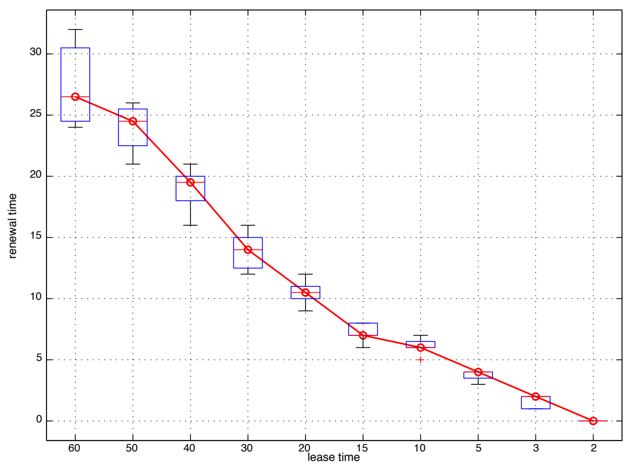


Figure 8: Lease renewal times

rates would be sufficient enough for successfully identifying the presence of our implemented covert channel. See Appendix A for a specific set of rules.

The *xid* implementation can be analysed using statistics or *entropy*. For a random string *xid*, let $H(xid)$ denote the entropy of *xid*. The entropy measures the information or surprise of the different values of *xid*. For a particular value, the surprise is $-log(P(xid))$. Thus, as it is well-known, $H(xid) = -\sum_i P(xid)log(P(xid))$ which basically means that when the *xid* field is filled with the start delimiter and clear bytes of the message to be transmitted, the entropy values obtained will differ notably with respect to random *xid* values. The only way of reverting this detection feature is "mixing" or introducing noise to the *xid* bytes. One way of doing this is by encrypting (but this means Alice and Bob need to share previously a short secret key) all the information to be coded.

As confidentiality is not a requirement (at least not permanently), we could send this short key, of a predefined size, before the (encrypted) start delimiter such that the DHCP server can check whether it is a packet conveying information. This change[12] in the information coding process will harden an *xid* field entropy analysis detection. Thus, the first packet sent by the DHCP client will include the following:

[12]Other implementation techniques using hash functions are possible.

$$Xid = k|E_k(DELIM) \tag{2}$$

In all cases, decreasing the leasing period would significantly increase the covert channel capacity, but at the same time it would increase the detection probability against global observers as defined in Section 3.1. In any case, both (non-)detectability and capacity will be improved if a compression method is used for the information to be sent (such as Huffman coding (Huffman, 1952)).

In summary, the *xid* implementation is stealthier than the other implemented methods because the original code fills the *xid* field with random data. On the other hand the *sname/file* method presents increased detectability. There are some features that might arise suspicion on an experienced observer: in many implementations these fields are filled with zeros when not used. Finally, the *options* method detectability might be improved by reducing the size of the option payload or by using different options to convey the data.

7. Conclusions

This paper presents an exhaustive analysis on the potential for covert communications in various short-range communication protocols. In particular, we study NetBIOS, Bluetooth, and DHCP because they are broadly used and have not been exploited for covert communications before. From the analysis we conclude that DHCP, a protocol that is extensively used for the configuration of IP-based devices, is the most suitable protocol to satisfy the requirements imposed by our scenario.

This results in the implementation of three storage channels that we integrate on a tool called HIDE_DHCP. These solutions present distinguishing features that makes them applicable to different circumstances. The main factors influencing the selection among the devised methods are the channel capacity and its detectability, which is leveraged by the fact that this is the first covert channel implementations based on DHCP. Moreover, the normal operation of the protocol is not altered by the subliminal channel.

Note that even when we use three fields for conveying information, the reached capacity is significantly smaller than in some previous covert channels. Nevertheless, a small covert channel capacity is not synonymous of lack of usability or importance because, as it is discussed by Moskowitz and Kang

(1994), in its small message criterion, the capacity is not important if the message to be transmitted is small.

Several information hiding techniques have been discussed for DHCP but only a few of them have been implemented. We have further analyzed the advantages and limitations as well as possible means of improvement. Our current work goes in this direction. Although we have performed some detectability tests with out-of-the-box intrusion detection solutions, much work can be done in this direction. We have constructed several basic Snort rules for detecting our implemented covert channel but a complete packet of rules for DHCP covert channels is to be provided (e.g. DHCP traffic rate detection).

We are taking the first steps towards porting our implementation into mobile scenarios. More importantly in recent mobile devices which integrates DHCP clients for wifi connections and existing malware apps, the existence of covert channels are a (theoretical) reality. This would increase the risk of information leakage for mobile users which greatly raises severe privacy risks.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Science and Innovation through the SPRINT (TIN2009-09237) and ARES (CSD2007-00004) projects, and by the Junta de Andalucia through the PISCIS (P10-TIC-06334) project. The SPRINT project is co-financed by FEDER (European Regional Development Fund) and the first author is supported by the Spanish Ministry of Education through the National F.P.U. Program.

Appendix A. Snort rules for detecting DHCP covert channels

In this section we show some rules which may be useful to signal the presence of covert channels on DHCP[13]. However, the main limitation is that these rules may lead to false positive cases. At least we have created a rule for each of the three different methods of covert channels. The first rule tries the detection on the default delimiter within the xid field:

```
alert udp $HOME_NET 68 -> any 67 (msg: "DHCP xid covert channel"; \
pcre: "/ (. | \ n) {4} [0-4] ts \ / / A"; sid: 1000000, rev : 1 ;)
```

[13] See Snort manual for rules writing in `http://manual.snort.org/`

This rule checks originated traffic in port 68 from a device in the local network addressed to any other device in port 67 (including 255.255.255.255), that is, originating from a client to a DHCP server. The payload of the packets must match the perl regular expression, defined with the keyword "pcre". This expression reads any 4 characters (. | \n) and then find a numeric value between 0 and 4 plus the delimiter, starting from the beginning of the payload. The user can change the delimiter value. In this case, these rules will not detect the channel presence.

The following rule searches for the *sname and file* method. In this case the size of the message to be sent, along with the delimiters, is shorter than the available size in both fields.

```
alert udp $HOMENET 68 -> any 67 (msg: "DHCP sname-file covert channel"; \
content: "| 00 | / st", offset: 44, depth: 4; \
content: "/ st"; within: 188, distance: 0; \
sid: 1000001, rev: 1 ;)
```

In this case the delimiter is sought after byte 44, which is when sname field begins. At this point it is checked whether the next four bytes correspond exactly to the null character followed by the delimiter, and if a match occurs, the next delimiter is searched in the 188 following bytes. The first delimiter detection may be sufficient due to the characteristics of the information usually contained in these fields. On the other hand, this rule will again not be useful if the user decides to create a new channel delimiter.

Lastly we present a rule that allows the detection of a DHCP covert channel which makes use of the option 224 (as described in Section5.3). In this particular case, the rule set identifies the channel traffic only in the case of a message larger than 255 bytes.

```
alert udp $HOMENET 68 -> any 67 (msg: "DHCP option covert channel"; \
content: "| ff e0 |", offset: 240; \
isdataat: 255, relative; sid: 1000002, rev: 1 ;)
```

This rule tries to find the tag and length; i.e., values 224 and 255 from the area corresponding to the options field. Besides, if the content is specified, from that position on must be at least 255 bytes of data so that the rule is triggered. As in previous cases, this rule is only useful if the option used for the channel creation is 224 (otherwise the channel will not be detected).

All these rules are useful in order to determine the existence of covert channels. In the first two methods, even if only the start delimiter arrives to the IDS the provided rules are triggered. The presence of a unique delimiter should not be reason enough to alert the user as there is the possibility, however remote, that an unmodified client generates, for instance, a random xid that matches the delimiter value being sought, thereby giving rise to a false positive. Consequently, in order to determine the existence of covert channels with certainty (i.e. the implemented covert channel evasion rate will be null), we need to use preprocesors (Stream5 UDP for session tracking). These preprocessors together with Snort features like *flowbits* provides the IDS with "memory", therefore being able to detect delimiters which arrive in different UDP packets. Also, Activate/Dynamic rules could be used in a similar way but still if the delimiters are dynamic the detection might be difficult.

References

Bluetooth SIG . Specification: Adopted Documents. online; 2012. Available 1 Nov. 2012.

Brand SL. Department of Defense Trusted Computer System Evaluation Criteria, "The Orange Book". Technical Report DoD 5200.28-STD; U.S. Department of Defense; 1985.

Cabuk S, Brodley CE, Shields C. IP Covert Timing Channels: Design and Detection. In: Proceedings of the 11th ACM Conference on Computer and Communications Security. New York, NY, USA: ACM Press; CCS'04; 2004. p. 178–87.

Cabuk S, Brodley CE, Shields C. IP Covert Channel Detection. ACM Trans Inf Syst Secur 2009;12:22:1–22:29.

Daemon9 . Loki2 (the implementation). 1997.

Droms R. RFC 2131 - Dynamic Host Configuration Protocols. 1997.

Dyatlov A. Firepass - Gray-World.net Team. 2003.

Gianvecchio S, Wang H. An entropy-based approach to detecting covert timing channels. IEEE Trans Dependable Secur Comput 2011;8:785–97.

Giffin J, Greenstadt R, Litwack P, Tibbetts R. Covert Messaging Through TCP Timestamps. In: Proceedings of the Privacy Enhancing Technologies Workshop (PET). 2002. p. 194–208.

Girling CG. Covert Channels in LAN‘s. IEEE Trans Software Eng 1987;13(2):292–6.

Gligor VD. A Guide to Understanding Covert Channel Analysis of Trusted Systems, “The Light Pink Book”. Technical Report NCSC-TG-030; U.S. National Computer Security Center; 1993.

Handel TG, Sandford MT. Hiding Data in the OSI Network Model. In: Proceedings of the First International Workshop on Information Hiding. London, UK: Springer-Verlag; 1996. p. 23–38.

Huffman D. A method for the construction of minimum-redundancy codes. Proceedings of the IRE 1952;40(9):1098 –101.

IEEE Computer Society . IEEE Std 802.15.1-2005, IEEE Standard for Information technology - Telecommunications and information exchange between systems - Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs). Technical Report; IEEE Computer Society; New York, NY, USA; 2005.

ISC . DHCP - Internet Systems Consortium, Inc. 2011.

Kaminsky D. Tunneling Audio, Video, SSH and pretty much anything else over DNS. 2004.

Lampson BW. A Note on the Confinement Problem. Commun ACM 1973;16(10):613–5.

Li S, Ephremides A. Covert channels in ad-hoc wireless networks. Ad Hoc Netw 2010;8:135–47.

Lucena NB, Lewandowski G, Chapin SJ. Covert Channels in IPv6. In: Privacy Enhancing Technologies. Berlin Heidelberg: Springer-Verlag; volume 3856 of *LNCS*; 2006. p. 147–66.

Luo X, Chan E, Chang R. Cloak: A ten-fold way for reliable covert communications. In: Biskup J, Lopez J, editors. Computer Security - ESORICS 2007. Springer Berlin / Heidelberg; volume 4734 of *LNCS*; 2007. p. 283–98.

McHugh J. Covert Channels Analysis: A Chapter of the Handbook for the Computer Security Certification of Trusted Systems. Technical Memorandum 5540:062A; Naval Research Laboratory; Washington, D.C.; 1996.

Moskowitz IS, Kang MH. Covert Channels – Here to Stay? In: Compass'94: 9th Annual Conference on Computer Assurance. Gaithersburg, MD: National Institute of Standards and Technology; 1994. p. 235–43.

NetBIOS Working Group . RFC 1001 - Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods. Technical Report; NetBIOS Working Group; 1987a.

NetBIOS Working Group . RFC 1002 - Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods. Technical Report; NetBIOS Working Group; 1987b.

Rowland CH. Covert Channels in the TCP/IP protocol suite. 1996.

Shah G, Molina A, Blaze M. Keyboards and Covert Channels. In: USENIX-SS'06: Proceedings of the 15th Conference on USENIX Security Symposium. Berkeley, CA, USA: USENIX Association; 2006. p. 59–75.

Simmons GJ. The Prisoners' Problem and the Subliminal Channel. In: Chaum D, editor. Advances in Cryptology: Proceedings of CRYPTO. Santa Barbara, California, USA: Plenum Press; LNCS; 1983. p. 51–67.

Snort . http://www.snort.org/. 2012.

Stødle D. Ping Tunnel - Send TCP traffic over ICMP. 2005.

Wolf M. Covert Channels in LAN Protocols. In: LANSEC '89: Proceedings on the Workshop for European Institute for System Security on Local Area Network Security. London, UK: Springer-Verlag; 1989. p. 91–101.

Wu J, Wang Y, Ding L, Liao X. Improving performance of network covert timing channel through huffman coding. Mathematical and Computer Modelling 2012;55(1-2):69–79.