# Introducing Dead Drops to Network Steganography using ARP-Caches and SNMP-Walks

**Tobias Schmidbauer**
University of Hagen
Germany

**Steffen Wendzel**
Worms University of Applied Science
& Fraunhofer FKIE
Germany

**Aleksandra Mileva**
University Goce Delcev
Macedonia

**Wojciech Mazurczyk**
Warsaw University of Technology
Poland

## ABSTRACT

Network covert channels enable various secret data exchange scenarios among two or more secret parties via a communication network. The diversity of the existing network covert channel techniques has rapidly increased due to research during the last couple of years and most of them share the same characteristics, i.e., they require a direct communication between the participating partners. However, it is sometimes simply not possible or it can raise suspicions to communicate directly. That is why, in this paper we introduce a new concept we call *"dead drop"*, i.e., a covert network storage which does not depend on the direct network traffic exchange between covert communication sides. Instead, the covert sender stores secret information in the ARP (*Address Resolution Protocol*) cache of an unaware host that is not involved in the hidden data exchange. Thus, the ARP cache is used as a covert network storage and the accumulated information can then be extracted by the covert receiver using SNMP (*Simple Network Management Protocol*).

## KEYWORDS

network covert channels, network steganography, covert communication, ARP, SNMP, dead drop

**ACM Reference Format:**
Tobias Schmidbauer, Steffen Wendzel, Aleksandra Mileva, and Wojciech Mazurczyk. 2019. Introducing Dead Drops to Network Steganography using ARP-Caches and SNMP-Walks. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019) (ARES '19), August 26–29, 2019, Canterbury, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3339252.3341488


*ARES '19, August 26–29, 2019, Canterbury, United Kingdom*

ACM ISBN 978-1-4503-7164-3/19/08…$15.00
https://doi.org/10.1145/3339252.3341488

## 1 INTRODUCTION

Covert channels (which are also called CC in this paper) are hidden communication channels that can be applied to operating systems, analog media and digital networks. Originally defined by Lampson as unforeseen communication channels [1] they are considered as policy-breaking mechanisms since the 1980's. In the networking context, covert channels are a sub-topic of network information hiding, where steganographic methods are utilized to create covert channels: a carrier, i.e., legitimate network traffic is used to conceal information, and the secrecy is typically based upon the concept of not being discovered [2], [3].

Currently, there are many existing methods to transport hidden information via covert channels. As Handel and Sandford already remarked in 1995, practically in every layer of the OSI reference model many different channels are known and can be further constructed [4]. A survey of different methods of covert channels can be found for example in [6, 7, 11].

In this paper, we assume a scenario which relies upon a "dead drop" used as a covert storage – the term is borrowed from the espionage tradecraft where such a technique was used to pass information between two spies using a secret location, thus not requiring them to meet directly. This drop allows to separate the covert sender (CS) and the covert receiver (CR) in a way that they do not interact directly with each other; they do not have to be active exactly at the same time, i.e., sending and receiving process can be de-coupled. This is similar to a scenario described in [5] for IoT steganography, where a smart building is used as a covert data storage. However, in this paper we do not rely on IoT components or IoT protocols but instead we utilize standard network protocols, especially Address Resolution Protocol (ARP). In comparison to the IoT protocols, ARP

is present in practically every Local Area Network (LAN). LANs in the typical enterprise environment can be of any size and they are typically used by a number of users which might not be allowed, due to a enforced security policy, to exchange data directly. In this paper, in order to enable covert communication that can bypass such a security-policy, we propose to exploit ARP caches (also called ARP tables). This type of hidden data exchange can be potentially more difficult to detect when compared to utilization of the information hiding techniques used, e.g., for e-mail-based data leakage, though ARP is limited to the collision domain.

Considering above it must be noted that performing hidden data exchange in the way proposed in this paper, i.e., using network covert storage, is significantly different from how typical network covert channels function. The vast majority of network steganography methods share the same characteristics [7, 11], i.e., they require a direct overt communication in which secrets are embedded between the covert sender and the covert receiver.

Moreover, they require that either both, the covert sender and the covert receiver, must be active simultaneously to embed/extract secrets or specially dedicated protocols are needed to achieve a successful hidden data exchange. Note, however, that in some cases fulfilling such requirements is not possible or it can raise too much attention which can contribute to easier uncovering of hidden communication. Thus, it must be noted that the approach proposed in this paper allows covert sender and receiver to be active at different times and only using standard network protocols.

Therefore, in this paper we make two key contributions:

- We show that network functionality deployed in all typical LAN hosts can be easily exploited as a *covert network storage*. Therefore, we present a novel and tailored hiding technique.
- We perform a proof-of-concept implementation of the proposed network covert storage-based method and analyze its capabilities experimentally (e.g., storage data size and storage duration).

As the data injection part of the covert channel can only be used in the same collision domain it can be used to transfer data from a well protected host without access to foreign networks like the internet to lesser protected hosts that are granted access to other networks. Furthermore it could be possible to extract information from an encapsulated host that is connected over VPN to a secure network environment. The connection has to be established over a device providing a collision domain to the encapsulated host and this device could be used as drop for hidden information, even if all upper layer traffic is blocked except over the encrypted VPN connection. There is no prove of concept for this scenario and this has to be tested in future work.

The rest of the paper is structured as follows. Sect. 2 introduces fundamentals and discusses related work. Sect. 3 describes the high-level concept of the proposed covert channel. Sect. 4 describes the implementation and performed experiments related especially to the number of entries that can be stored, the retention period in caches, extracting hidden data and at least a few countermeasures that can be performed. The conclusion of this paper is made in Sect. 5.

## 2 FUNDAMENTALS & RELATED WORK

We first explain the fundamentals of the LAN communication protocols that we exploit for our steganographic data storage. Afterwards, we cover related work.

### Utilized Communication Protocols

There are two protocols used for the covert channel proposed in this paper: ARP is utilized to write data into a remote host's ARP cache. SNMP is utilized to read the entries that are stored in the host's ARP cache. In the following subsections, both protocols are discussed, especially their features used for the covert communication purposes.

*Address Resolution Protocol.* ARP was defined in RFC 826 [15] in 1982 by David C. Plummer and is applied in today's networks to map the Internet Protocol (IP) address to the appropriate hardware address (called MAC address or physical address) of a network interface card (NIC). A system that only knows the IP address of another system has to create an ARP request to resolve its MAC address. As shown in Fig. 1, Host A only knows the IP of Host X and creates the ARP request in step a). System A sends the request in step b) to all hosts within the subnet using the broadcast destination MAC address. The requested Host X identifies its own protocol address and performs the following steps as shown in c): X adds the (MAC, IP) tuple to its own ARP table or updates a differing existing entry of A. Also, X reworks the request in step d) to an ARP reply and adds its own MAC address as it was requested by Host A. Next, X then sends the reply directly via unicast to the MAC address of Host A, which is shown in step e). In the last step f), Host A adds the (MAC, IP) tuple to its own ARP table. The described steps are all limited to a collision domain.

The ARP table is a local passive cache that stores known (MAC, IP) tuples of the devices that have already communicated with its owner. Because the ARP request is broadcasted on the LAN, all hosts on that network will have updated the sender's hardware address in their table if its IP address was already in the table.

Starting from Windows Vista, ARP caching behavior has been changed to comply with RFC 4861 [16] (Neighbor Discovery Protocol for IP version 6 (IPv6)) and the IPv6 neighbor

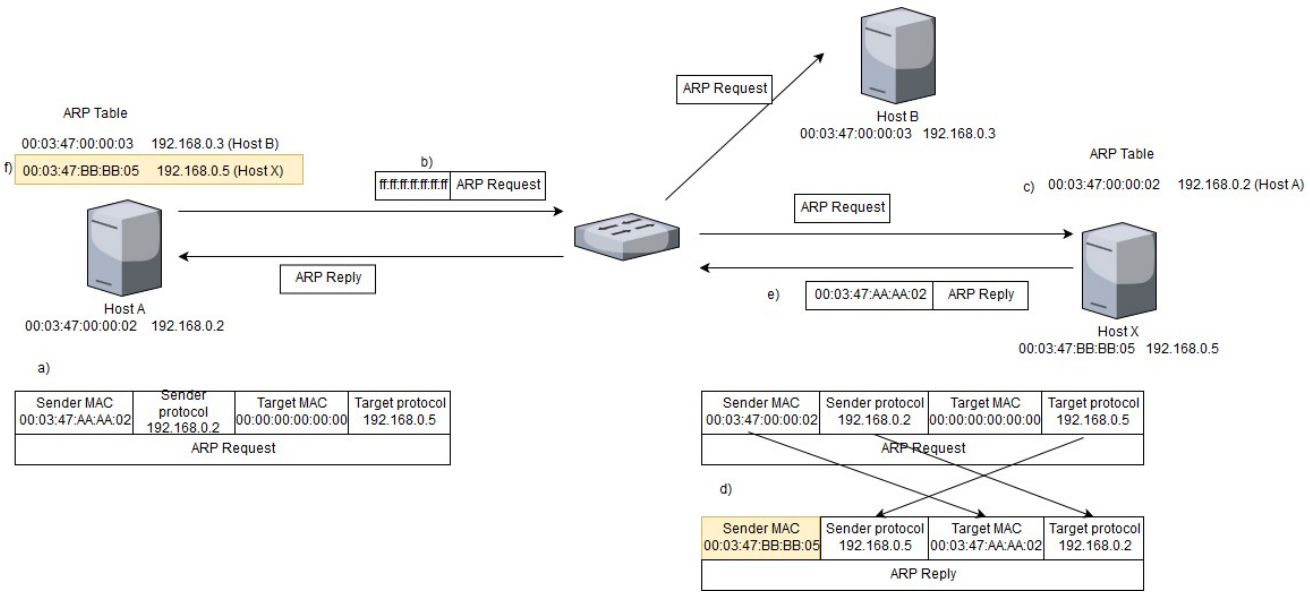


**Figure 1: ARP request-reply cycle**

discovery process. An ARP cache entry for IPv4 is an example of a neighbor cache entry, and after its creation, it is in a "reachable" state. So, an ARP age timeout threshold is in fact the *Reachable Time* in Windows, which is calculated using the following formula

$$\text{Reachable Time} = \text{BRT} \cdot (\text{Random value between MIN\_RANDOM\_COUNT and MAX\_RANDOM\_COUNT}),$$

where BRT (*Base Reachable Time*) is 30,000 ms (30 s), and MIN_RANDOM_COUNT and MAX_RANDOM_COUNT are 0.5 and 1.5, respectively.

From this formula, the Reachable Time, or ARP age timeout threshold is between 15 and 45 seconds. One can change the Base Reachable Time, using the `netsh` command. Additionally, the maximal default ARP cache size in Windows corresponds to the Neighbor Cache Limit and it is equal to 256 entries per interface. It can be changed by setting a new value for the option `neighborcachelimit` using `netsh`.

*Simple Network Management Protocol.* The SNMP protocol was defined in RFC 1157 [17] and is the most-widely used standard protocol for reading information about configurations and status of managed networking devices. There are two versions of SNMP available that are used in LANs. Version 2c (SNMPv2) that is defined in RFC 1901 [18] uses community strings to grant access to so-called *Object Identifiers (OID)*. Because there is no encryption implemented in this version, a string, that is equal to a password, has to be transferred as plaintext over the network and can be discovered by any host. To solve this issue, SNMP Version 3 (SNMPv3), defined in the RFCs 3410 to 3418 [19–27], implements authentication and encryption to transfer data more safely between the communicating SNMP partners. Also SNMPv3 provides a new mechanism to grant users access to the specific information. Both protocol versions have in common that a so called management host can request information from the other SNMP-enabled devices by polling information. This communication is based upon UDP, requesting a specific *Object Identifier (OID)*. Those OIDs are organized in a tree structure, a so-called *Structure of Management Information (SMI)*. All of those Objects, including the ARP cache, can be polled by the management server.

In [9], a measurement study was conducted where the usage of the different SNMP versions is presented. In 2007, version 3, which at this point was already defined as standard and available for 5 years, was not used in the analyzed traces. Though the version 1 and 2 of SNMP that were already declared as "historic" still were used in 2007. This indicates that the introduction of this version was significantly postponed, thus after over 15 years there is still evidence of encountering networks where SNMPv2 is still utilized. Version 2 is still supported in modern OS and has not been fully replaced by SNMPv3.

## Related Work

During several last decades, many network covert channels have been proposed and they have been already surveyed

| Trace | Hours | Messages | Traffic Size (MB) | Message/min. | Traffic Size/min. (MB) |
|---|---|---|---|---|---|
| 101t02 | 162.98 | 51772136 | 6963 | 5294.32 | 0.712 |
| 101t05 | 336.00 | 40072529 | 14043 | 1987.72 | 0.697 |
| 102t01 | 294.62 | 258010521 | 77789 | 14595.67 | 4.401 |
| 103t02 | 159,21 | 1871361365 | 130858 | 91217.19 | 13.699 |
| 104t01 | 4.00 | 15099 | 10 | 62.91 | 0.042 |
| 105t01 | 580.60 | 25298667 | 2898 | 726.22 | 0.083 |
| 106t01 | 22.08 | 89277889 | 24683 | 67389.71 | 18.631 |
| 112t01 | 208.02 | 2619884 | 312 | 209.91 | 0.025 |
| **Overall** | 1767.51 | 1338428090 | 257556 | 12620.66 | 2.429 |

**Table 1: Analyzed SNMP traffic [9]**

from different perspective (see, e.g., [4, 6–8, 10–13] and references therein). Thus, in this section, we will focus only on presenting existing works that are utilizing either ARP, SNMP or dead drop-like functioning. The ARP protocol was used as a carrier for covert channels in [10] by encoding information in the target IP field of the protocol. The hidden information is encoded in the last $w$ bits ($4 \leq w < 8$) of the last byte of the target IP address. To distinguish the ARP requests belonging to the covert channel, the first $8 - w$ bits of the last byte encodes the sending second, first inverted and then XORed with the first $8 - w$ bits of the covert $w$ bits. Jankowski et al. [13] proposed an inter-protocol steganographic method named *PadSteg*, which utilizes ARP and other protocols (TCP or ICMP) together with an Etherleak vulnerability (improper Ethernet frame padding) to provide a secret communication between members of the hidden groups in LANs. Each node uses the ARP request together with frame padding bits for advertising its presence to other members of the hidden group.

The idea of providing a network-based covert storage cache is a recent concept and it has been first introduced in [5] in the context of IoT devices. Using two different IoT protocols, secrets could be stored in the unused registers of IoT sensors or in actuator states, e.g., by slightly modulating the heating level of a heater in an automated building.

Indirect network-based storage covert channels over innocent intermediate hosts have been presented in several works. In 1997, Rowland [29] suggested a so called "bounce" channel, in which the sender encodes secret information in the ISN of a TCP SYN packet and sends it to an intermediate host (with a spoofed IP source address set to the IP address of the receiver). The intermediate host responds with a TCP SYN/ACK or SYN/RST packet, in which the Acknowledgment number field is set to ISN+1. A similar idea, but with a message encoded in the payload of the ICMP Echo Request packet sent to an intermediate host is used by Zelenchuk [30]. Danezis [28] uses an intermediate host with a weak implementation of the TCP/IP stack, in which the Identification field of IPv4 packets is generated by a simple counter. Covert sender and covert receiver use the ICMP Echo Request/Response mechanism or the congestion control features of TCP to modulate the Identification counter in the intermediate host. The covert message is encoded in different modulation rates. Another indirect covert channel uses the DNS functionality of negative domain name caching [31]. The covert communication parties first agree on an ordered list of non-existent domain names, and then the covert receiver issues recursive queries to an intermediate DNS server (for all domains in the list that correspond to the binary 1). Several covert channel receivers can then obtain the hidden message by issuing non-recursive queries to the same DNS server for the entire list and checking which domains are in the list, and which are not.

## 3 PROPOSED APPROACH

We propose to utilize the ARP caches in LANs as storage caches for covert data. A covert sender influences the ARP cache of a third-party host to place the hidden data while a covert receiver queries the ARP cache of the third-party host to retrieve the hidden data. In the following subsections, we will first describe the assumed communication scenario. Then we will explain the process of secret data bit encoding for the proposed covert channel. Finally, we outline the process of extracting secrets from the ARP caches using SNMP.

### Communication Scenario

The communication scenario assumed in this paper significantly differs from the common network covert channels scenario mainly because no direct connection between the Covert Sender (CS) and the Covert Receiver (CR) is required (Fig. 2). As illustrated there, the CS possesses secret information that it wants to store in the network-accessible ARP cache (a). Thus, he exploits the ARP cache of a third-party system by sending a fake ARP request containing a (MAC,IP) tuple (b). The actual host reflected by the (MAC,IP) tuple does not exist and represents encoded secret information.

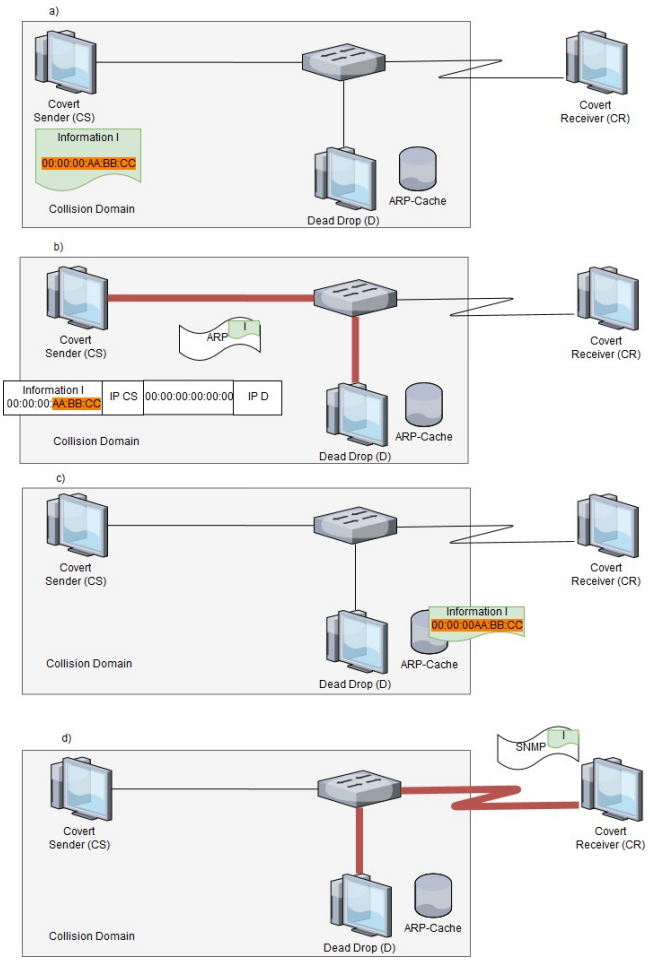


**Figure 2: ARP-based communications scenario**

The third-party host Dead Drop (D) adds the tuple to its local ARP cache (c) where it can be requested by the CR (d) for some time depending on the lifetime of ARP cache entries on the third-party host.

CR needs to be allowed to request data from D with the use of SNMP protocol. There are two possible scenarios for CR using SNMP:

- CR is allowed to poll via SNMP,
- CR is not allowed to poll via SNMP, but knows the Community String.

In the first case, CR is a SNMP management host that is allowed to generate SNMP traffic. In this case, to transport information over the covert channel there is a need to have control over this server. This allows the usage of SNMPv3 with encryption and user management as CR is the management host. In the second case, CR has listened to the communication of legitimate SNMP communication and extracted the community string or figured out the correct community

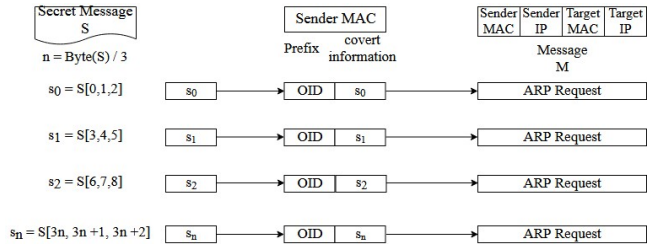


**Figure 3: Secret data encoding in the last three bytes of the Sender-MAC Address**

string by guessing it. This can happen for example when the default group "public" stays enabled in the configuration. This second case only can be used if SNMP traffic is not encrypted, like in SNMPv2 for example.

## Secret Information Encoding

As shown in Sect. 2, the sender IP address and the sender MAC address are the only utilized ARP header fields for the proposed covert channel. The contained information are stored in a targeted host's ARP cache. The CS can manipulate both fields to encode secret data.

The IP address is more regulated in a LAN because of specific subnets and private IP ranges. To be inconspicuous, the IP addresses need to be close to existent addresses which appear in the network, so it can only be used to manage the sequence of information. To this end, we propose that CS first of all analysis the network traffic in order not to collide with existing addresses. The range of IPs that are not existent can be used for the CC. CR also needs to analyze the traffic of the network to find out which entries in the cache belong to the hidden communication. The addresses that are used by existing devices in passing by network traffic can be dropped.

So the MAC address is used to encode the secret information as it is less regulated and the IP address is used to number the secret data consecutively. The MAC address as information carrier consists of six bytes of which three bytes are reserved for the ID of the manufacturer of the network card, a so called *Organizationally Unique Identifier (OUI)*. So only the last three bytes are available for the secret data $S$. Our encoding is shown in Fig. 3.

The Secret Message $S$ is split into parts that fit the utilizable space of the ARP packet. The length of $S$ must be divisible by three, which is the number of bytes that can be sent within one ARP request. If this is not the case, $S$ will be padded with the value 0x0A, which represents a line feed, until it is a multiple of 3. Different to [10], we propose to add a hexadecimal number to each character which is known in advance to the CS and CR as a pre-shared secret. The advantage of such approach is that the number can change for each

separate hidden message. As shown in Fig. 3, three bytes are grouped to the secret messages $s_0$ to $s_2$ which are three-byte subsets of $S$.The first three bytes of the Sender MAC are filled with a prefix that equates to the OUIs that are used by devices in the collision domain. For higher covertness, more than one prefix might be used.

### Secret Data Extraction

To extract the data from the ARP cache of the Dead Drop, SNMPv2 was set up on D and CR as clarified in Fig. 2, cf. Sect. 3. The focus is on proving that the described concept is feasible thus in this communication scenario both systems were aware of the utilized community string. CR was able to read all entries in the cache of D by using the command `snmpwalk` with the correct community string and the OID that represents the information that is stored in the ARP cache.

## 4 EXPERIMENTAL EVALUATION

We implemented our covert channel as a Python script, also taking advantage of the `Scapy` library for generating packets. The experiments with the above described methods were repeated on several operating systems to ensure the observed behavior is not specific to one OS. As Dead Drops we used two popular Linux distributions, namely OpenSuSE 42.3 and Ubuntu 18.04. Also Windows 7 and Windows 10 were tested as they are popular in enterprise environments. The insertion and the extraction of entries have been performed by another system that was running OpenSuSE 42.3. All systems were set up as virtual hosts using *VirtualBox* 6.0.4.

In the following subsections, we first specify which settings were used during the experiments. Next, we describe the caching behaviour of the tested systems, especially when many ARP cache entries are stored. Afterwards, we describe the retention period of stored entries followed by a description of the observations that were made reading information from the caches. Next, we present results related to the detectability of the proposed approach and finally, we discuss potential countermeasures.

### Implementation

We decided to use the subnet 192.168.3.0/24 which allows us to add 254 hosts to our environment. The virtual router was given the first IP address of the subnet, and the test systems were configured to use the following addresses 192.168.3.1 to 192.168.3.6, so they were not available for use by the covert channel (as they are in use by real devices). As mentioned before, we propose to scan the network for existing addresses to avoid collisions. In our test environment, this is not a problem so we decided to implement the CC using the IP addresses starting from 192.168.3.128. The pre-shared secret added as padding character was chosen to be 0x2A.

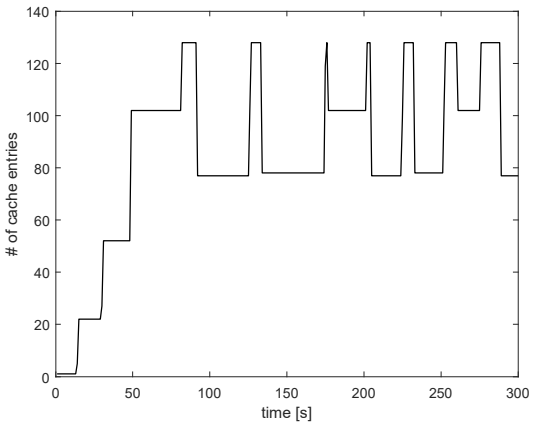


**Figure 4: Ubuntu caching behavior**

### Storing Hidden Data

To evaluate how the ARP caches of the different operating systems behave, we have performed a set of experiments. First, we tried to determine how many entries are stored in a cache without losing entries to replacement algorithms. We inserted as many entries as possible in order to observe the caches using local shell scripts. The performance experiment showed that on both Linux systems there was a maximum of 127 entries after which the cached entries were cut and cleared by replacement algorithms. This was confirmed by a second test that tried to add entries near the assumed maximum for Ubuntu (Fig. 4) and for OpenSuSE (Fig. 5).

The expected maximum of 127 entries was confirmed with slowly nearing to this number of entries. First of all we tried to reach the critical number of entries in order to observe if the assumed maximum is correct. Fig. 4 confirms this for the Ubuntu system. First 20 entries were added to the cache, then 30, and again 50 new tuples to reach the number of 100 stored addresses. Next, the number of 127 entries was exceeded. At this point the number of cached entries dropped by the number of 50. Those entries were the first 50 that were added in two steps, the others remained untouched. By exceeding 127 entries next time, the oldest 50 entries which were added by step 3 were dropped. As the maximum number was reached a third time, the last added entries again were dropped and the newer ones endured. This behavior was forced a few times and allowed us to conclude that always the oldest entries are dropped and the newer ones are preserved.

Nearly the same behavior was experienced by the OpenSuSE system as visualized in Fig. 5. As with the Ubuntu System the maximum of 127 entries could be reached several times without dropping any entries. When we added

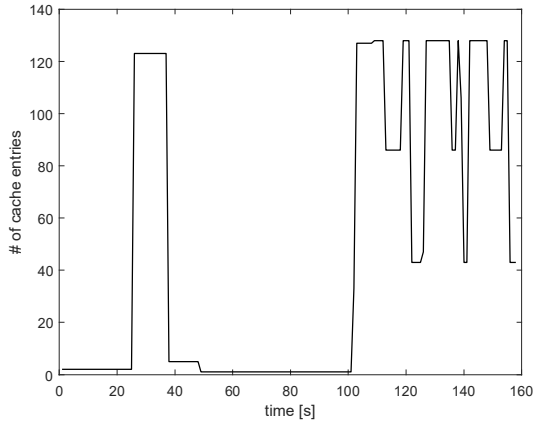


**Figure 5: OpenSuSE caching behavior**

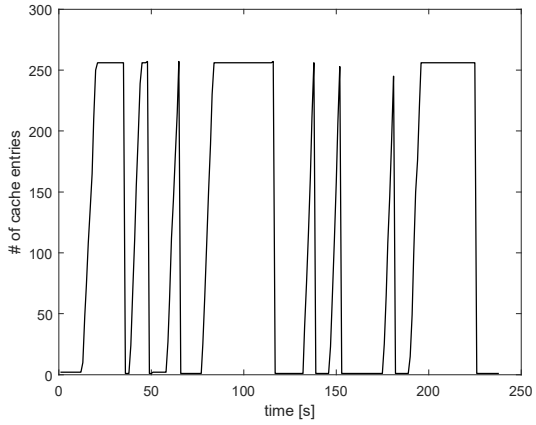


**Figure 7: Windows 7 caching behavior**

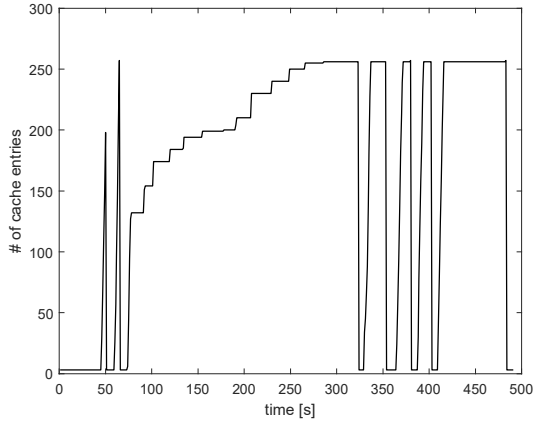


**Figure 6: Windows 10 caching behavior**

one more entry all of them were removed except a certain number of entries. Exceeding 127 entries always caused the oldest entries to be dropped.

The same approach was implemented for the Windows machines and a first test on both systems, Windows 7 and Windows 10, indicated a maximum number of 256 entries as expected (cf. Sect. 2).

As the first two peaks in Fig. 6 illustrate, the replacement algorithm in Windows is much faster than in Linux. Within even a second, the entries were removed from the cache by reaching the critical number of entries. Next, we tried to reach the maximum by adding few entries step by step, until the maximum number of 256. By adding one more entry, the cache was fully cleared. As opposed to Linux, no entries remained in the cache, even some default multicast addresses that were resident before were dropped. As shown in this figure, such a behavior was repeatable.

Also Windows 7 behaved the same way. As in case of Windows 10 all entries including the resident multicast addresses that were stored at the beginning of the experiment were dropped by exceeding 256 entries (Fig. 7).

### Retention Period

The retention period of addresses in the ARP cache of Dead Drops is important to create and maintain our covert channel. A long residing time of information in cache allows the resulting covert channel to be less detectable as it would be possible to send fewer ARP requests per minute. Furthermore the loss of information in the cache will lead to a broken covert message as we supposed this to happen after short time (see Sect. 2).

In this test scenario, we tried to add a number of entries near to the maximums determined in the previous section. After adding these entries we observed the period of time when the entries were residing in the cache without being refreshed as shown in Fig. 8.

Without refreshing, all ARP cache entries remained for several days. The experiment has been canceled after 9 days of constantly showing the same number of tuples. For the Windows 10 system, a little peak on day 2 has been experienced caused by a parallel test, that added 2 more entries.The Windows OSs cached 250 addresses that were related to the retention test, the Linux Systems 120. None of them were dropped during the observation time. This means that the covert data storage can easily survive multiple days, decoupling the activity periods of CS and CR.

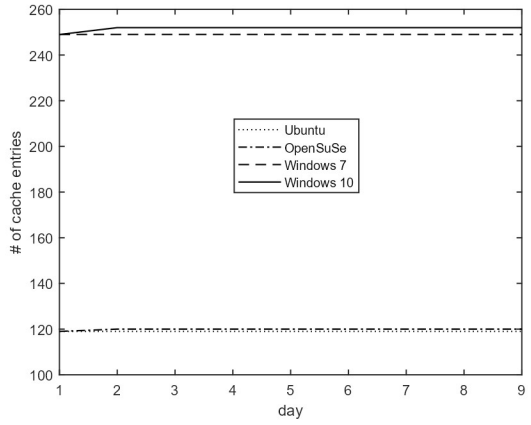


**Figure 8: Duration of the retention by days without refreshing**

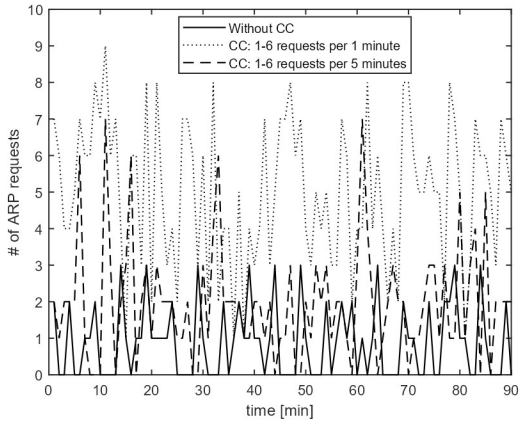


**Figure 9: ARP requests with 2 active machines**

The observations made differ from the results that have been expected within the described functionality of the Address Resolution Protocol (Sect. 2) that implements an aging mechanism for all entries in the ARP cache. The result of the retention experiment denies this expectation as it proves that entries are stored over a much more longer time periods without being refreshed by additional constant ARP requests.

### Extraction

To test the extraction of entries from the ARP cache, we first rebooted the Dead Drop and then stored two password hashes from the Covert Sender's `/etc/passwd` file and a *hello world* message. We were able to extract all of the test messages from all of the Dead Drops by the Covert Receiver using the procedure described in Sect. 3. The separation of entries not belonging to the CC were not a problem as the IP addresses of the existing hosts were already known in advance. To decode the secret content correctly, we had to consider that there extracted strings of OIDs in Windows and Linux OSs have different formats.

### Detectability

Minimizing the detectability is one of the most important features of a covert channel. To be inconspicuous, the covert channel traffic characteristics has to be as similar as possible to the legitimate traffic. To evaluate the proposed method's detectability, we monitored the ARP requests in a LAN with 2 and 5 active machines, respectively. In addition to measuring the requests without covert channels to observe the natural ARP noise of a network, we decided to investigate two test scenarios with active covert channels. In the first setup a random number between 1 and 6 ARP requests is sent per minute. This essentially re-creates the experiment described in [10] (Sect. V and Sect. VI). As the experimental environment is not the same, it must be noted that the results may differ from the original setup used in the mentioned paper. The second scenario is based on a timespan of 5 minutes wherein between 1 and 6 ARP requests are randomly sent. The measurements with 2 hosts were realized using Windows 7 and OpenSuSE 42.3 OS-based machines. For the scenario with 5 hosts we used one Windows 7, one Windows 10, one Ubuntu 18.04 and two OpenSuSE 42.3 systems. The ARP requests belonging to the covert channel were always sent from a OpenSuSE system to the Windows 7 host.

Fig. 9 shows the observations made with two hosts. The network behaviour with the covert channel containing 1-6 ARP requests per minute differs clearly from a network within active hidden traffic. The number of requests per minute significantly and constantly stick out above the normal network behaviour. However, it must be noted that the covert channel with the request frequency of 1-6 requests per 5 minutes aligns more to the natural behaviour of a LAN, disregarding a few peaks that are highly above the normal number of ARP requests. Those peaks count up to 7 requests per minute for all active hosts in the network while in the state without an active covert channel the number of requests never exceeds 3 per minute. This suspicious anomalies could be resolved by adding a maximum number of covert channel requests sent within one minute.

Fig. 10 illustrates the observations made for the scenario where there are 5 active hosts. The first observation that can be made is that the natural ARP noise of a network is higher than with fewer hosts, which is not a surprise due to more talking network members. The raised quantity

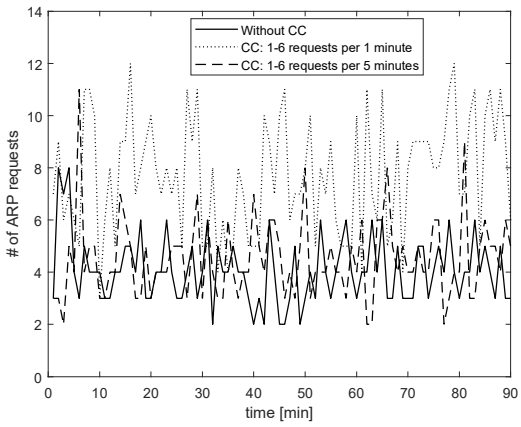


**Figure 10: ARP requests with 5 active machines**

of requests per minute allows active hidden channels to be more inconspicious. This can be seen by comparing the covert channel with the frequency of 1-6 requests per minute to the measure without CC. The number of ARP requests per minute with covert channel is now at some points on or under the natural level, though the traffic is still clear to notice. This lets us conclude, that the more hosts are members of a collision domain the lesser the evidence of a covert channel is. This assumption is strengthened by the measurement of the covert channel with the frequency of 1-6 ARP requests per 5 minutes. The number of requests blends well with the natural noise, so the hidden traffic can not be differed from it anymore. Except of some anomalies where the number of requests per minute lies high over a normal level the amount is constantly at the level without covert channel or even under it.

The observation made leads to the conclusion that the bandwidth of the injection part of our covert channel grows with the number of active hosts in a collision domain. The bandwidth for the frequency of 1-6 requests per minute allows to transfer from 180 byte/hour to 1080 byte/hour (on average 630 byte/hour), while lower frequent channel of 1-6 requests per 5 minutes allows to transfer secret data from 36 byte/hour to 216 byte/hour (on average 126 byte/hour). The 5 minute channel is plausible in a environment containing 5 hosts as shown in this subsection.

We also analyzed how much traffic is caused by reading the ARP cache with SNMP. A cache on a Windows system contains as shown in Sect. 4 up to 256 entries. So we filled the cache of the Windows 10 machine to this maximum and polled the cache. This caused the number of 524 SNMP packages containing 47,968 byte of data. Compared to the measurements created in [9] that can be found in Table. 1 in Sect. 2. The conspicuousness of the requested ARP cache differs by the noise caused by the regular SNMP traffic. In the network of trace 103t02, the number of messages is higher than 90,000 per minute causing more than 13.000kB/minute traffic so the reading part of our covert channel would be challenging to discover. In contrast to this, the trace 112t01 for example only has 200 messages per minute transferred that contain only 25kB/Minute. Additional 500 Messages would result in 47kB of transferred data, causing a peak that clearly could be discovered. This may be solved with only polling a few entries of the cache which can be implemented by using the sub OIDs of the ARP cache.

### Potential Countermeasures

A number of countermeasures is imaginable to prevent or detect our proposed covert storage cache and its utilization.

First of all, the capacity of the ARP cache (number of entries that can be stored) could be reduced in most network environments. Especially, the Windows host limit of 256 entries appears to be too high and could be reduced to, e.g., 64 entries. Also, the cached entries should be deleted after a certain time. This does not happen as shown in the retention period experiment. Even if not all entries are deleted due to performance issues there should be random entries renewed after a few minutes as this does not affect the performance of the network too much. As shown by [10], entries for really existing hosts are renewed permanently.

Another starting point can be the monitoring of ARP caches to detect an abnormal amount (or abnormal increase) of entries using simple heuristics on the basis of NIDS rules (signatures). This is easier than detecting suspicious ARP requests that could belong to the covert channel. Those methods can be combined, for example into a blind clearing of random entries after a certain amount of tuples in a cache is reached, e.g., on the basis of the Least Recently Used pace replacement algorithm. Another method is trying to ping the IPs that are cached randomly. If there are several unreachable devices, this could indicate the presence of a the covert storage. However, it could also mean that ICMP echo requests are filtered.

Another option is to use classic port/address locking: if an attacker connects to a network via an unused network port (e.g., at a switch), he can only access the network if the port is not locked [14]. Port/address locking allows only specific hosts an access to pre-defined ports. However, MAC spoofing could circumvent this protection technique. Currently unused network ports could moreover be deactivated. Ultimately, Physical Access Control (PAC) could be an additional (physical) layer of security that only allows authenticated individuals to access rooms where switch ports are accessible.

However, also SNMP could be subject to countermeasures. The need of reading the entries via `snmpwalk` can indicate

that an Dead Drop exists. If no encryption is used, an active warden can check tuples that were transferred are also existent. Using SNMPv3 with encryption avoids reading information by hosts that eavesdrop the community string but causes the problem that the management host could extract tuples over an encrypted channel.

## 5 CONCLUSION

We have shown how the commonly used network protocols ARP and SNMP can be utilized to establish a covert network storage cache. The cache can be filled by a sender and read by a receiver. The filling and reading can be separated by a duration of several days and the utilization of suitable coding enables a robust extraction of data, though it is only possible to fill the cache within a collision domain. To the best of our knowledge, this is the first general-purpose covert data cache for LAN environments. The bandwidth that can be established for covert transmissions while remaining stealthy grows with the number of active systems. Even in environments that run a low number of hosts, the covert channel is viable as shown in the 2-system setup with the 5 minute approach.

In future work, we plan to evaluate capabilities of the different network covert storage realizations that would be created with the use other protocols.

## ACKNOWLEDGMENTS

The authors would like to thank Prof. Jörg Keller for his help and useful comments related to this research.

## REFERENCES

[1] B. W. Lampson: "A note on the confinement problem," Commun. ACM, Vol. 16(10), pp. 613–615, 1973.

[2] E. Zielinska, W. Mazurczyk, and K. Szczypiorski: "Trends in Steganography," Commun. of ACM, vol. 57, no. 3, pp. 86–95, 2014.

[3] S. Wendzel, W. Mazurczyk, L. Caviglione, M. Meier: "Hidden and Uncontrolled On the Emergence of Network Steganography," In: Proc. of ISSE 2014 Securing Electronic Business Processes, pp. 123-133, Springer Vieweg, 2014

[4] T. Handel and M. Sandford: "Hiding Data in the OSI Network Model," in Proc. of: Information Hiding Wksh., Cambridge, U.K., 1996, pp. 23–38.

[5] S. Wendzel, W. Mazurczyk, and G. Haas: "Steganography for cyber-physical systems," Journal of Cyber Security and Mobility, no. 6.2, pp. 105–126, 2017.

[6] S. Wendzel, S. Zander, B. Fechner, C. Herdin: "Pattern-based Survey and Categorization of Network covert channels," ACM Computing Surveys (CSUR), no. 47(3), pp. 50:1–50:26, 2015.

[7] S. Zander, G. Grenville, P. Branch: "A survey of covert channels and countermeasures in computer network protocols," IEEE Communications Surveys & Tutorials, Vol. 9(3), pp. 44-57, IEEE, 2007.

[8] L. Caviglione, M. Podolski, W. Mazurczyk and M. Ianigro, "Covert Channels in Personal Cloud Storage Services: The Case of Dropbox," IEEE Trans. on Industrial Informatics, vol. 13, no. 4, pp. 1921-1931, 2017.

[9] J. Schönwälder, A. Pras, M. Harvan, J. Schippers, and R. van de Meent: "SNMP Traffic Analysis: Approaches, Tools, and First Results," in 10th IFIP/IEEE International Symposium on Integrated Network Management, Munich, Germany, 2007, pp. 323–332.

[10] L. Ji, Y. Fan, and C. Ma: "Covert channel for local area network," in Proc. of the IEEE Int. Conf. on Wireless Communications, Networking and Information Security, WCNIS 2010, Beijing, China, 2010, pp. 316–319.

[11] W. Mazurczyk, S. Wendzel, S. Zander et al.: "Information Hiding in Communication Networks," Wiley-IEEE, 2016.

[12] A. Mileva and B. Panajotov: "Covert Channels in the TCP/IP Protocol Stack – Ext. Ver.," Central European Journal of Computer Science, 2014.

[13] B. Jankowski, W. Mazurczyk, K. Szczypiorski: "PadSteg: Introducing Inter-Protocol Steganography," Telecommunication Systems, Vol. 52(2), pp. 1101–1111, 2013.

[14] A. Guruprasad, P. Pandey, B. Prashant: "Security Features in Ethernet Switches for Access Networks," in Proc. Conf. on Convergent Techn. for the Asia-Pacific Region (TENCON 2003), Vol. 3, pp. 1211-1214, 2003.

[15] D. C. Plummer: "An Ethernet Address Resolution Protocol; or Converting Network Protocol Addresses to 48.bit Eth. Address for Transmission on Ethernet Hardware," RFC 826, Network Working Group, 1982.

[16] T. Narten, E. Nordmark, W. Simpson, H. Soliman: "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861 , Network Working Group, 2007.

[17] J. Case, M. Fedor, M. Schoffstall, J. Davin: "A Simple Network Management Protocol (SNMP)," RFC1157, Network Working Group, 1990.

[18] J. Case, J. McCloghrie, M. Rose, S. Waldbusser: " Introduction to Community-based SNMPv2, " RFC1901, Network Working Group, 1996.

[19] J. Case, R. Mundy, D. Partain, B. Stewart: "Introduction and Applicability Statements for Internet Standard Management Framework" RFC3410, Network Working Group, 2002.

[20] D. Harrington, R. Persuhn, B. Wijnen: "An Architecture for Describing SNMP Management Frameworks," RFC3411, Net. Working Group, 2002.

[21] J. Case, D. Harrington, R. Persuhn, B. Wijnen: "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)," RFC3412, Network Working Group, 2002.

[22] D. Levi, P. Meyer, B. Stewart: "Simple Network Management Protocol (SNMP) Applications," RFC3413, Network Working Group, 2002.

[23] U. Blumenthal, B. Wijnen: " User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)," RFC3414, Network Working Group, 2002.

[24] B. Wijnen, R. Presuhn, K. McCloghire: "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)," RFC3415, Network Working Group, 2002.

[25] R. Presuhn, J. Case, J. McCloghrie, M. Rose, S. Waldbusser: "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)," RFC3416, Network Working Group, 2002.

[26] R. Presuhn, J. Case, J. McCloghrie, M. Rose, S. Waldbusser: " Transport Mappings for the Simple Network Management Protocol (SNMP)," RFC3417, Network Working Group, 2002.

[27] R. Presuhn, J. Case, J. McCloghrie, M. Rose, S. Waldbusser: " Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)," RFC3418, Network Working Group, 2002.

[28] G. Danezis: "Covert Communications Despite Traffic Data Retention". In: Security Protocols XVI, LNCS vol. 6615, 2008.

[29] C. H. Rowland: "Covert Channels in the TCP/IP Protocol Suite," First Monday, Vol. 2(5), July 1997, https://firstmonday.org/ojs/index.php/fm/article/view/528/449.

[30] I. Zelenchuk: "Skeeve – ICMP Bounce Tunnel," 2004, http://www.gray-world.net/poc_skeeve.shtml.

[31] Anonymous: "DNS Covert Channels and Bouncing Techniques," 2005, https://seclists.org/fulldisclosure/2005/Jul/att-452/p63_dns_worm_covert_channel.txt.