Preventing Protocol Switching Covert Channels

Steffen Wendzel[1,2] and Jörg Keller[1]
[1]*Faculty of Mathematics and Computer Science, University of Hagen, Germany*
[2]*Department of Computer Science, Augsburg University of Applied Sciences, Germany*
steffen.wendzel@hs-augsburg.de, joerg.keller@fernuni-hagen.de

***Abstract*—Network covert channels enable a policy-breaking network communication (e.g., within botnets). Within the last years, new covert channel techniques arose which are based on the capability of protocol switching. Such protocol switching covert channels operate within overlay networks and can (as a special case) contain their own internal control protocols. We present the first approach to effectively limit the bitrate of such covert channels by introducing a new active warden. We present a calculation method for the maximum usable bitrate of these channels in case the active warden is used. We discuss implementation details of the active warden and discuss results from experiments that indicate the usability in practice. Additionally, we present means to enhance the practical application of our active warden by applying a formal grammar-based whitelisting and by proposing the combination of a previously developed detection technique in combination with our presented approach.**

***Keywords*-Protocol Switching Covert Channel; Protocol Channel; Active Warden; Covert Channel Detection; Network Security**

I. Introduction

This work is an extended version of [1] in which we introduced the design and implementation of an active warden countering protocol switching covert channels in a basic version with a constant delay.

The term *covert channel* refers to a type of channel defined as *not intended for information transfer* by Lampson in 1973 [2]. The goal of using covert channels is to transfer information without raising attention while breaking a security policy [3]. Covert channels have been a focus of research for decades. In addition to Lampson's work, the topic was described in [4] and [5]. While covert channels can also occur on local systems, we focus on covert channels within computer networks.

Covert channels are basically divided into two classes: covert storage channels and covert timing channels [4]. To transfer hidden information, a covert storage channel alters storage attributes (e.g., values of a network packet's header) and a timing channel alters timing behavior (e.g., the timing of network packets) [6].

A well-known technology to counter covert channels is the active warden, i.e., a system counteracting a covert channel communication. While passive wardens monitor and report events (e.g., for intrusion detection), active wardens (e.g., traffic normalizers [7]) are capable of modifying network traffic [8] to prevent steganographic information transfer.

Recently, the capability to keep a low profile resulted in a raising importance of network covert channels because of their use cases. For instance, covert channels can be used to control botnets in a hidden way [9]. Covert channel techniques can also be used by journalists to transfer illicit information, i.e., they can generally contribute to the free expression of opinions [6].

A first covert channel able to switch a network protocol based on a user's command called *LOKI2* was presented in 1997 [10]. Within the last decade, different new covert channel techniques occurred and not all of them were already addressed by protection means. These new techniques enable covert channels to switch their communication protocol automatically and transparently. They are enabled to cooperate in overlay networks by using internal control protocols as presented in [11]. Since covert channels are a dual-use good, these novel techniques do also enrich the security of botnets.

We focus on two new covert channel techniques, *protocol hopping covert channels* (PHCC) as well as called *protocol channels* (PCs). Both channels build the family of protocol switching covert channels since they rely on the capability to switch their utilized network protocols.

PHCC were presented in [12] and were improved in [11]. These channels transfer hidden information using several network protocols to raise as little attention as possible due to peculiar protocol behavior, i.e., they combine several single-protocol network storage covert channels. For instance, a simple PHCC could use the "User-Agent" field in HTTP as well as the message number in POP3 "RETR" (retrieve) requests to transfer hidden information. Protocol switches in a PHCC are transparent for the covert channel's user and the user utilizes the channel as a black-box that handles the splitting of the payload in data chunks sent using the different protocols. To work properly, the PHCC must be able to preserve the order of network packets across the different protocols.

PCs were introduced in [13] and signal information solely by transferring network protocols of a pre-defined set in an order that represents hidden information. Protocols are therefore linked to secret values, e.g., a HTTP packet could represent the value "1" and a POP3 packet could represent

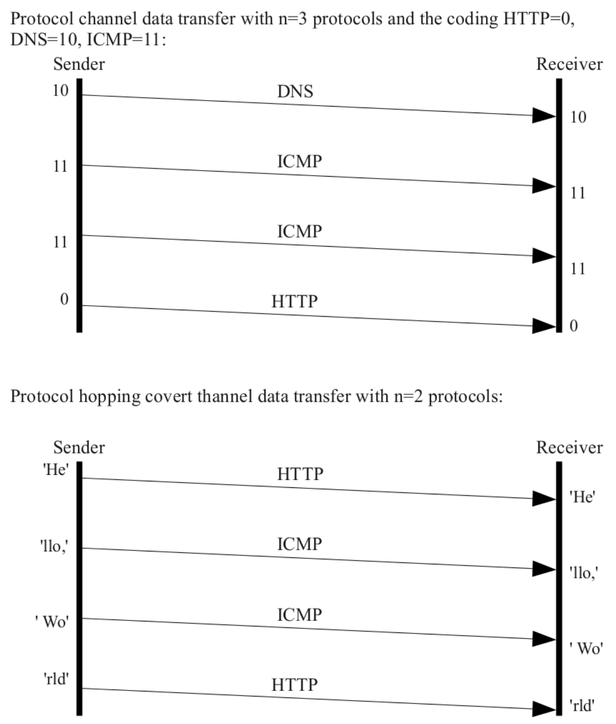
Protocol channel data transfer with n=3 protocols and the coding HTTP=0, DNS=10, ICMP=11:
Sender
Receiver
10
DNS
10
11
ICMP
11
11
ICMP
11
0
HTTP
0
Protocol hopping covert thannel data transfer with n=2 protocols:
Sender
Receiver
'He'
HTTP
'He'
'llo,'
ICMP
'llo,'
' Wo'
ICMP
' Wo'
'rld'
HTTP
'rld'

Figure 1. Comparison of a PC and a PHCC.

the value “0”. To transfer the message “110” via this PC, the sender is required to send two HTTP packets followed by a POP3 packet. The bitrate of a PC is usually limited to a few hundred bit/s. However, for an attacker this is fast enough to transfer passwords, selected records or tweets.

Figure 1 compares both, the PC and the PHCC concept. The difference between both channels is, that PCs signal their hidden information solely by the use of a selected network protocol in a network packet, while PHCCs combine multiple covert storage channels simultaneously and place hidden information in arbitrary storage locations of these channels.

We present the first concept as well as an implementation of an active warden able to significantly reduce the bitrate of both, PHCC and PC. While we do not present a universal solution countering all covert channels, this is the first work discussing means against PHCC and PC. The limitation of these advanced covert channels is more challenging in comparison to single-protocol covert channels. We evaluate our results via realistic experiments and we demonstrate that our approach is useful for the practical operation in organizations. Due to these achievements, our active warden decreases the attractiveness of both channel types for attackers.

The remainder of this paper is structured as follows. Section II provides an overview of related work in the area of covert channels. Section III introduces the idea and theory of our active warden, while Section IV discusses our implementation and our experimental results, and Section V focuses on the practical aspects of the presented approach and discusses improvements for covert channels to bypass the active warden. We propose improvements for the practical use of our approach by applying a formal grammar in Section VI and by combining our active warden with a previously developed detection capability for PCs in Section VII. We conclude in Section VIII.

II. Related Work

Various techniques for embedding covert channels in network packet data and its timing were developed within the last decades. For instance, Girling and Wolf were the first authors to create network covert channels by modifying LAN frames [14], [15] and Rowland initially presented covert channels for IP and TCP [16]. Rutkowska discovered the idea of a passive covert channel that does not actively generate own traffic but piggibacks regular traffic of a system’s users by modifying the TCP Initial Sequence Number (ISN) [17]. Therefore, Rutkowska introduced a translation layer for ISN values into the Linux kernel. Cabuk et al. developed a technique to embedd hidden information in the timing of network packets [18], while Ahsan modified the order of network packets to achieve the same goal [19]. Besides, covert channel presence was discussed in the DHCP protocol [20], in IPv6 [21], in additional areas of TCP (e.g., in TCP timestamps [22]) and IPv4 (e.g., by alternations of fragment sizes [23]), and in business processes [24].

Means were developed to deal with the problem of covert channels, like the pump [25], which is a device that limits the number of acknowledgement messages from a higher to a lower security level and thus affects covert timing channels based on ACKs; a concept extended in [26]. Other well-known techniques are, for instance, covert flow trees [27], which can be used to detect direct and indirect covert channels in source code, as well as the shared resource matrix (SRM) methodology [28] and the extended SRM [29], which can also be applied to source code but can additionally be used in other steps of the software development lifecycle. A newer apprach is program transformation to remove timing leaks and covert timing channels [30], and steganalysis of covert channels in VoIP traffic [31]. Hu introduced fuzzy time to limit the capacity of timing channels between virtual machines on the VAX security kernel [32], and Zander et al. as well as Berk et al. applied different means based on machine learning and statistics to detect covert timing channels in network transmissions [6], [33].

A concept similar to PCs is the idea of a port knocking covert channel as presented by deGraaf et al. [34]. Since port knocking alters information specifying the encapsulated protocol (using the UDP destination port), a port knocking covert channel can be considered as variant of a PC. A first detection algorithm for PC (but not for PHCC) was presented

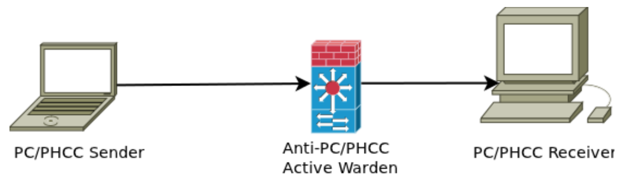

Figure 2. Location of the Anti-PC/PHCC active warden

in [35], but we are not aware of any research to limit the bitrate of PC and PHCC or to prevent them.

PHCCs were also used by Yarochkin et al. to create adaptable network covert channels with the goal to change the utilized network protocols, if required [36]. For instance, a covert channel could use HTTP and ICMP. Due to a change in the firewall configuration between covert channel sender and covert channel receiver, the ICMP protocol get might blocked. The idea of Yarochkin et al. is to filter out such blocked protocols and afterwards switch to another protocol to overcome the firewall. Li and He presented a similar approach based on the concept of natural selection in which survival values are calculated for each network protocol to evaluate its usefulness [37]. The authors only utilize protocols with the currently highest survival values to overcome firewalls.

III. An Active Warden Countering Protocol Switching Covert Channels

The idea of an active warden addressing PHCC and PC focuses on one aspect both covert channel types share: the protocol switches. For both channel types, it is a necessity to ensure that network packets using different network protocols reach their destination in the same order as they were sent. Our approach of an active warden for countering PHCC and PC monitors the protocol switching behavior of network hosts and introduces delays in network packets if a protocol switch occurs.

Like the network pump (a device that only allows acknowledgement messages from a high to a low system in a regulated manner [25]), we have no explicit detection capabilities in our active warden but aim on limiting the channel's bitrate nevertheless. In comparison to the pump, we do not limit acknowledgement messages but alter protocol occurrences. Using the presented technique, we can prevent performance decreases for downloads and uploads and minimize practical implications (cf. Section V-A) by installing the active warden on a company's uplink or between (autonomous) systems. The warden only affects those network packets that change the last occurring network protocol.

The active warden must be located between a covert channel sender and a covert channel receiver (Figure 2). To prevent PC/PHCC-based data leakage for enterprises, a suitable location would be the company's uplink to the Internet. The active warden's delay is configurable and thus makes our approach adjustable, i.e., an administrator can choose the individual optimum between maximized protection and maximized throughput. Formally, this creates a multi-criterion optimization problem. For a given delay d, data leakage can occur at a maximum rate $R(d)$, that is decreasing with increasing d. On the other hand, the side-effects for legitimate users will increase with increasing d, i.e., can be modeled by a function $S(d)$. Ideally, one would like to minimize both, R and S, which is however not possible. One can combine the two in a target function

$$\text{penalty}(d) = \epsilon \cdot R(d) + (1 - \epsilon) \cdot S(d) ,$$

which is then to be minimized, i.e., after fixing a suitable $\epsilon \in [0;1]$ the minimization results in an optimal d that represents the administrator's priorities. As both R and S are assumed to be monotonous, a Pareto optimum can be found in the sense that a further reduction of R by increasing d cannot be achieved without increasing S. Typically, instead of using R and S directly, they are normalized to a certain range such as interval $[0;1]$, and they might be adapted by linear or non-linear functions that reflect e.g., the severeness of an increased leakage.

Imagine a PC using ICMP (1 bit) and UDP (0 bit) and the goal to transfer the message "0110001". In this case, the sender would need to send UDP, ICMP, ICMP, UDP, UDP, UDP, ICMP. If our active warden is located on a gateway between both hosts and can delay packets, which probably belong to a PC or PHCC, the successful information transfer will be corrupted. At the beginning, the sender sends an UDP packet, which is forwarded by the active warden. Afterwards, the sender sends the ICMP packet, which is delayed for a time d because a protocol switch happened. The next packet is an ICMP packet again and therefore not delayed but forwarded. Afterwards an UDP packet occurs, which is delayed for a time d, too. The next two UDP packets do not change the last protocol and are therefore forwarded. The last ICMP packet results in an additional packet switch and is therefore delayed for time d again. If d is 1 second, then all delayed packets will arrive after the non-delayed packets if the sender did not introduce synthetic delays itself. The resulting packet order at the receiver's side will be UDP, ICMP, UDP, UDP, ICMP, UDP, ICMP, or "01**00**101" (containing two incorrect bit values).

The situation is similar for PHCC where the hidden information is not represented through the protocol itself but through alternations of a protocol's attributes (such as the IPv4 TTL or the HTTP "User-Agent"). If the active warden modifies PHCC transmissions sent via different protocols, the reassembled payload will be jumbled.

In order to prevent the covert channel users to take advantage of learning about the value of d, it might also be randomized, cf. Sections III-B and IV-B.

A. Bitrate Calculation

Tsai and Gligor introduced the formula $B = b \cdot (T_R + T_S + 2 \cdot T_{CS})^{-1}$ for calculating the bandwidth of covert channels using the values T_R (time to receive a covert message), T_S (time to send a covert message), T_{CS} (time required for the context switch between processes), and b (amount of transferred information per message) [38]. While the formula addresses local covert storage channels, all parameters are adaptable to a network covert channel as well.

We use a modification of this formula (cf. formula 1) to calculate the max. possible error-free bitrate of a PC in case our active warden is located between sender and receiver. We introduce the active warden's delay d multiplied with the probability p of a protocol switch per packet. Instead of T_R, T_S and T_{CS}, we use T to represent the *transmission* or *gap* time:

We call T the gap time to represent the minimal time difference between two packets of the covert channel. If T is too small, packets can overrun each other or the receiver cannot process them fast enough, thus, T is limited due to the technical environment of the channel: For high performance computers connected via gigabit links, T is small, while it will be bigger for systems connected via DSL over the Internet.

On the other hand, T can be understood as transmission time if the channel is designed to only transfer packets in a sequential manner, i.e., one packet has to be received before another packet can be sent. In that case, the gap time is equal to the transmission time. The transmission time is the time required for receiving and sending a packet including the processing time required by the active warden.

In the remainder, we call T the gap time, but as explained, T can also be the transmission time between packets since T depends on the implementation of the covert channel (if the channel is not capable of sending a successing packet before an earlier packet was received).

The amount of transferred data per packet (b) is 1 bit/packet in case two protocols are used for a PC since $b = \log_2 n$, where n is the number of protocols used. Thus, p as well as n will increase if more than two protocols are used (more information can be transferred per packet but the switching rate will also increase). In case of a PHCC, b depends on the amount of storage data per packet and not on the number of protocols used. Therefore, p will increase if more protocols are used but since the number of protocols is not linked to the amount of information transferred per packet, b will *not* increase if more protocols are used.

$$B = b \cdot (p \cdot d + T)^{-1} \qquad (1)$$

Theoretically, p is 0.5 if randomized input, a uniform coding and a set of two protocols is used since the next packet is either using the same protocol as the last (no protocol switch) or the other protocol (a protocol switch is taking place). In our experiments, the average protocol switching value for a typical protocol switching covert channel using only two protocols was $p = 0.4738806$. Thus, to transfer information without risking a corruption through a delay, a PC/PHCC user is forced to send packets with protocol switches in a way that the delay d cannot corrupt the packet order.

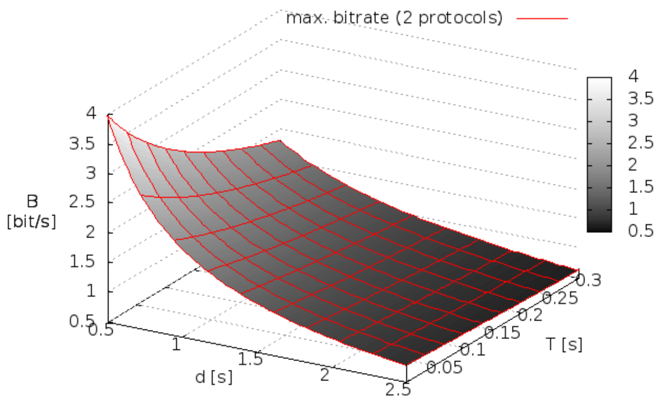

Figure 3. A PC's bitrate (B) using a set of two protocols depending on the delay d and the transmission time T.

As mentioned earlier, the value p depends on the amount of protocols used as well as on the channel's coding. If a uniform coding was used (as with optimized channels) and if two protocols are used p is approx. 0.5. In case four protocols are used, p is approx. 0.75. In general, for n protocols used p is $(1 - 1/n)$. Thus, formula 1 can be modified to the following version:

$$B = b \cdot ((1 - 1/n) \cdot d + T)^{-1} \qquad (2)$$

Protocol Channel: As also already discussed, $b = \log_2 n$ in case of a PC. Thus, $B = \log_2 n \cdot ((1 - 1/n) \cdot d + T)^{-1}$. Taking d as well as T into account using formula 1, we calculated the maximum useful bitrates for an uncorrupted PC transfer using a set of two protocols (Figure 3). According to our calculations, a PC's bitrate can be reduced to less than 1 bit/s if the active warden introduces a delay of 2.0 sec for protocol switches. For T, we used a time range obtained from measurements of the original "pct" program.

Protocol Hopping Covert Channel: For a PHCC, the parameter b varies more than parameter T (is, as in the case of a PC, usually very low). Therefore, we created a different plot where we set T to the static value 0.005, which we measured in experiments. Figure 4 shows the bitrate of a PHCC dependent on the delay d as well as the number b of transferred bits per packet. Obviously, the result of the

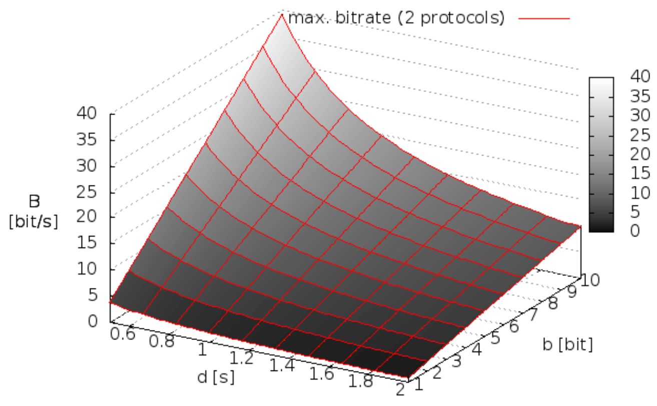

Figure 4. A PHCC's bitrate using *two* protocols, $T = 0.005$ and delays between 0.5 and 2s as well as the capability to transfer between 1 and 10 bits per packet.

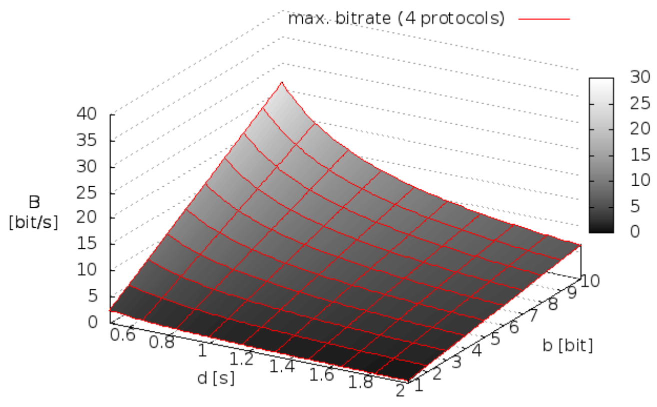

Figure 5. A PHCC's bitrate using *four* protocols, $T = 0.005$ and delays between 0.5 and 2 as well as the capability to transfer between 1 and 10 bits per packet.

PHCC is equal to the result of a PC if $b = 1$. However, PHCCs can carry more information and are therefore harder to limit, i.e., they require higher delays.

As shown in Figure 5, the bitrate of a PHCC decreases if the number of protocols used increases, since more protocol switches ($p = 1 - 1/4$) occur, i.e., the active warden's efficiency for PHCCs is positively correlated with p.

Before we experimentally validate the estimated bitrate of the protocol switching covert channel, we will discuss details of our implementation.

B. Improved Coding

The presented approach addresses PC and PHCC without special coding but with a coding as available in their respective proof-of-concept codes. We highlight the advances of a better coding for both, PC and PHCC, as a means to counter the active warden.

PC with improved coding: If a PC uses a coding that requires to send new packets only if a value unequal to the current value is required to be transferred, it can overcome the active warden, if sender and receiver are synchronized. This is possible if the sender only transfers a network packet if a protocol switch occurs, i.e., two packets of the same protocol are never transferred after another. The timing intervals between the protocol switches represent the amount of bits to transfer. Thus, such a covert channel would be a hybrid version of a timing channel and a PC.
Example: The sender sends a packet of protocol P_1. The active warden delays it for time d and forwards it. Three more bits as represented by P_1 shall be transferred. Therefore, the sender waits for three time slots. Afterwards, a bit represented by P_2 shall be transferred and the sender sends one such a packet. The active warden delays the packet for d and forwards it. If the sender sends P_1 again, it will also be delayed for d. The receiver will receive P_1, three waiting slots, P_2, P_1, i.e., the same input as was sent by the sender. The only disadvantage introduced by the active warden is the delay of d for all packets but this is a minor consequence for the covert channel since even if the message is delayed, it still reaches its receiver without comprising errors. Thus, such a coding would primarily result in side-effets on legitimate traffic due to d but would have no direct affect on the covert channel traffic.

To overcome this problem, an improved version of the active warden was developed. In our previous experiments, we focused on an active warden with a constant delay d. If d varies from packet to packet, or in other words, d is randomized (e.g., $d \in [0.1; 2]$ sec), previous packets are likely to overrun newer ones if the timing interval of the sender is too small. Thus, the sender is forced to use big waiting times and thus, will be forced to decrease its bitrate.

Besides the previously mentioned coding, a PC could also use other codings to improve the amount of bits transferred per protocol switch (b/p). For the default PC coding and two protocols, $p = 0.5$, but when a run length limited (RLL) coding (as used for hard disks [39]) is implemented, p can be decreased.

In case of geometrically distributed symbols, an optimized coding (Huffman coding) can help the covert channel's user to minimize the amount of packets to transfer, but – as usual for covert channel research – we focus on an optimized coding using a uniform distribution (e.g., the covert channel is used to transfer encrypted input).

Another variant of a PC might use unary encoding of symbols with $n = 2$ protocols. In order to send a value $i \in \{0, \ldots, k-1\}$, $i+1$ packets of the first protocol and $k-i$ packets of the second protocol are sent. Assuming that k is a power of two, then $b = \log_2(k)/(k+1)$ bits are transferred per packet, as k different symbols could be encoded in $\log_2(k)$ bits, and $k+1$ packets are used to transmit one symbol. Note that $b < 1$, for example with $k = 16$, we have $b < 0.25$. The sender then waits for some time before sending the packets for another symbol. The receiver can decode a symbol if the packets for one symbol arrive before any packets for the next symbol arrive. The sender might also use the two protocols alternately to code symbols. In both cases, the gap between two symbols must be rather large to overcome an active warden with a randomized delay d. Because the delay is unknown to the sender, it can neither rely on a maximum delay nor a minimum delay of any packet. Hence, to clearly separate the packets for successive symbols on the receiver side, the sender's waiting time between symbols must be high, at least larger than the maximum value of d. Thus, the bitrate of such a channel is restricted to b/d, which is less than 1 bit/s if $d \geq 1$ s as $b < 1$ bit/packet.

Receiver-side re-calculation attack for PCs: In case a constant delay is used, it is possible for the attacker to recalculate the original sequence of received packets of a PC. However, due to the jitter, it is possible that the attacker is forced to use error-correcting codes. The active warden can implement the previously mentioned randomized d to overcome this problem. Also, the overhead for error correction additionally reduces the available PC bitrate.

PHCC with internal control protocols A drawback of our approach is linked to a feature only available for PHCCs but not for PCs. PHCCs provide usually enough covert space to contain a covert channel-internal control protocol (called a *micro protocol*) [11]. Such micro protocols can be used to embed sequence numbers in covert channel packets as done in [12], [40], [41].

Using these sequence numbers, the receiver can reassemble network packets even if their ordering was disturbed [12]. While the active warden is not able to completely solve this problem, it *forces* a PHCC user to *use* a sequence number. Such a storage channel-internal sequence number usually consists of only 2-3 bits and thus, the active warden can break the receiver-side sorting nevertheless, if d is large enough.

Additionally, since these channels do only provide a few bits per packet, the active warden decreases the available space per packet by forcing the presence of a sequence number or of a larger presence value. Thus, a user is forced to send more packets to transfer the same amount of data than in the case where no active warden would be located on the path between PHCC sender and PHCC receiver.

However, it is important that the size of the micro protocol header is as tiny as possible since the few available bits of space provided within a PHCC should be used to transfer payload. If less space is available per packet due to a larger micro protocol header, more packets are required to be transferred to send a given payload to the receiver.

IV. Experimental Validation

In the following, we discuss our implementation and the results of our validation. We set up an experimental environment in form of a virtual network to represent a realistic scenario for a data leakage in an enterprise network. The content used in the experiment was generated by two available proof of concept tools that could be used by any attacker and thus, also represent realistic user generated traffic.

A. Implementation

For our test implementation, we set up a virtual network between two virtualized Linux 3.0 systems (a covert channel sender as well as a receiver with a local active warden instance) using *VirtualBox* (*www.virtualbox.org*). Both hosts were connected using a virtual Ethernet interface and IPv4. Our proof of concept code focused on layer 4 protocols identifiable by the IPv4 "protocol" field only. Therefore, we modified the *protocol channel tool* (pct) [42] that used ARP and ICMP to use UDP and ICMP instead. Additionally, we implemented the functionality to generate randomized input and to adjust the channel's bitrate.

To implement the network delays on the receiver system that is acting as both the active warden gateway and the protocol switching covert channel receiver at the same time, we made use of the firewall system *netfilter/iptables*. Netfilter/iptables provides a "queue" feature, which can be used to redirect data packets to a userspace program. Berrange implemented the Perl-based program *delay-net* [43] that enforces configurable network delays using the *IPQueue* module [44], which is based on the iptables queue feature. We modified delay-net in a way that it only delays packets after a protocol switch happened. We also implemented a third program to evaluate the correct transmission at the receiver's side, to test our prediction from formula 1 (cf. Section III-A).

Since this test focuses on the protocol switching capability of both, the PC and the PHCC, at the same time, the testing method is valid for both covert channel types. However, we additionally used the *protocol hopping covert channel tool* (phcct) [45] to verify the results for PHCC with and without micro protocols and high data rates since PCs cannot comprise such internal control protocols.

B. Results

In our test configuration, the value T is quite small (we measured 0.005 in average) in comparison to the delay time d. As mentioned in the previous Section, we were able to determine $p = 0.4738806$ through observing the behavior

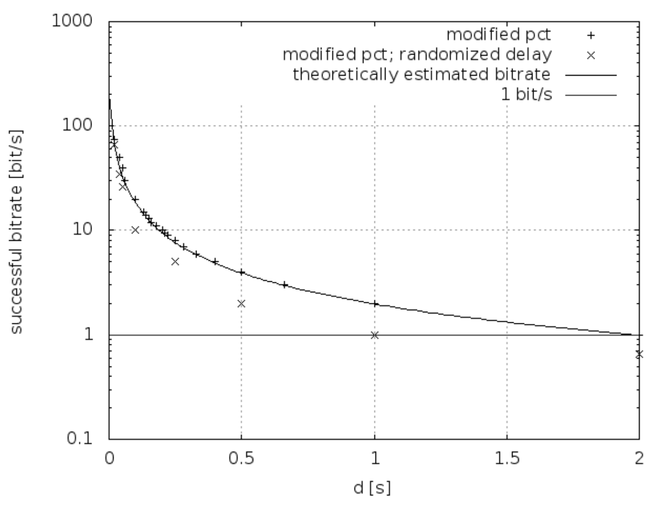

Figure 6. Measured maximal bitrate of the modified *pct* dependent on the active warden's delay in case a constant or a randomized delay is applied. In comparison, we show formula 1 using the estimated protocol switching value.

of the modified *pct* in our virtual test network. However, p turned out to be slightly higher (0.53) in a real network environment, where protocols occur, which do not belong to the PC, and therefore result in few additional protocol switches.

In our test setting we sent PC data using different bitrates and monitored the correctness of the received packet order at the receiver's side. Using this method, we were able to find out the maximum bitrate able to work error-free dependent on the delay introduced by the active warden. Figure 6 shows the results in comparison to our calculation of B (formula 1). The differences between B and our recorded values are small. We evaluated both, an active warden with a constant delay d as well as an active warden with a randomized delay in the range $[0, d[$.

An active warden with a *constant* delay value $d = 2.1\ s$ ($T = 0.005$) reduces the bitrate limit required for a successful transmission of data to $1\ bit/s$. If $d = 1.0\ s$, the bitrate limit is reduced to a maximum of $2.088\ bit/s$.

If we apply a *randomized* delay, the results are better than in case of a constant delay. To reduce the bitrate to $1\ bit/s$ we only needed to apply a max. delay of $d = 1\ s$. A max. delay of $d = 2\ s$ reduces the PC's bitrate to $0.65\ bit/s$. The higher efficiency of a randomized delay comes due to the circumstance, that when a packet is delayed and the following packet is delayed as well, the second packet can be forwarded earlier than the first packet and thus, jumbles the packet sequence. If both packets are delayed with a constant time, a later delayed packet cannot overrun a previously delayed packet. Figure 7 visualizes both delay differences.

Comparing both results as shown in Figure 6, we identified the randomized delay to provide a better efficiency than

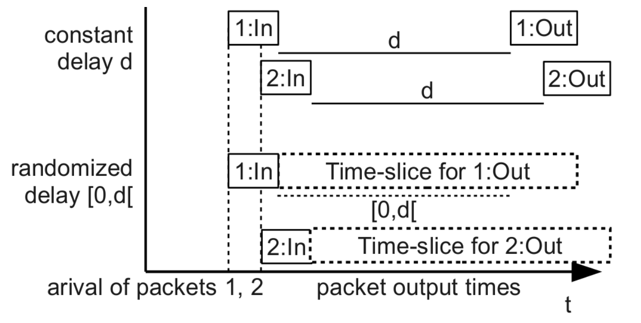

Figure 7. Output of two delayed packets for a constant and a randomized delay.

the constant delay.

Since PC and PHCC can both be seen as covert storage channels, the interesting aspect was to test PHCC situations, which are not exactly the same as for PCs. We therefore ran two experiments:

In the *first experiment*, phcct was used to transfer data with its capability to re-sort jumbled packet sequences using its internal micro protocol. The results showed, that phcct was indeed capable of re-sorting test traffic for 10KByte and 100kByte payload transfers (each sent 10 times through the active warden).

For the *second experiment*, we modified phcct in a way that the re-sorting capability of the internal control protocol was turned off. Since phcct is capable of transferring $b = 792$ bits per packet, the applied delays are less efficient than with a smaller b (as it is given for PCs). However, no transfer without errors was possible (*with and without* the active warden), if phcct was ran and transferred more than 1KByte (11 packets) of payload. Thus, high bitrate transmissions – as done by phcct – without a re-sorting capability are practically not feasible for PHCCs.

V. Discussion

After the concept, implementation and the measured results of the active warden were presented, this Section aims on discussing practical aspects of the active warden and improvements for covert channels to counter the active warden.

A. Practical Aspects

A goal of the presented active warden approach is to design the system for a practical use. The requirement for only small delays is – even if a user's initial request to a website will be delayed – an acceptable limitation for legitimate traffic in high-security environments since delays of only around 2s can reduce the useful bitrate of PCs to a maximum of 1 bit/s. For PHCC, the value can differ if the channel provides high values for b. However, to achieve the goal of practical usefulness, it is necessary to implement additional functionality because of the following reasons:

DNS requests: Typically, a user sends DNS requests to a DNS server and, after receiving a response, connects to a system using another protocol. It is required to take care of this typical effect (and similar effects such as using HTTPS right after a user clicked on a link of a HTTP-based website). Protocol switches occur in both cases (DNS→HTTP and HTTP→HTTPS, respectively). The DNS server and the system a user connects to (usually a web server) will in almost all cases have different addresses, thus it is easy to address this problem if the active warden distinguishes destination hosts. In Section VI, we present an approach based on formal grammar to address this problem.

Different protocols on a single host: However, situations are thinkable in which a user is connected to one host using two different protocols, e.g., to an SMTP server and an IMAP server hosted on the same system. In such cases, whitelisting (e.g., defining trusted hosts) can reduce this problem. This problem is also addressed in Section VI.

Multiple senders: In an enterprise network, there are usually a number of different computers with Internet access. If the active warden is located on the uplink, it will notice many protocol switches since different systems use different protocols to achieve different tasks. The active warden should distinguish source addresses to solve this problem. Therefore, it is also necessary to distinguish source addresses in the active warden to overcome this problem.

Some companies run a *network address translation* (NAT) service *within* their network. These systems would appear as a single system to the active warden although the systems as well as the active warden are located inside the company network. Thus, the NAT'ed systems would face delays. A whitelisting is no sufficient solution since these address translated systems are required to be protected from data leakage too. A possible remedy for that problem would be to use *remote physical device fingerprinting* [46] to count the number of NAT'ed systems and apply smaller delays per packet switch if the number of hosts behind NAT increases (and vice versa).

Redundancy: As all normalizer and firewall-like systems, our prototype of an active warden can result in a single point of failure if not operated on a redundant installation. Modifications of existing redundancy protocols (e.g., the common address redundancy protocol, CARP) might be used to solve this problem. However, as any firewall-like system, the active warden introduces side-effects, i.e., delays, into network traffic.

End-User Limitation: As explained, the effect for end-users is low if the active warden is used.

While an extensive end-user study was not part of this work, we measured different HTTP request-response times for 10MByte downloads over the active warden. The standard download time in our network was 0.41-0.57s. After we installed the active warden, we ran a HTTP download as well as a 0.25 bit/s PC to simulate a number of protocol switches as they occur for modern websites (multiple DNS requests for a whole site are normal since they can include script sources from other domains). This increased the download time to 0.43-3s. We observed that the basic limitation for connections happens in the establishment phase where a new protocol (HTTP over TCP) occurred. In the context of the 4s-rule for website rendering [47], we can assume that our active warden is valuable for practical use-cases.

To summarize, all mentioned problems (except the network address translation) are solvable by adding the mentioned simple features. The configurable delay parameter d provides administrators a way to adjust the efficiency of the active warden to their requirements. Since only protocol switching packets are affected by the active warden, most of a network's traffic is not affected, i.e., download rates and upload rates will not decrease notably.

B. Covert Channel Improvements

While we already discussed improved encoding techniques for both, PC and PHCC, in Section III-B, this Section will highlight architectural means, which can be used by both channel types to bypass the active warden. We will also discuss the subject of covert channel-internal control protocols for PHCC.

Multiple Covert Channel Senders: While the previously mentioned approach of distinguishing source addresses is a requirement for the practical application of the active warden, a distributed covert channel sender can take advantage of it: If the covert channel sender consists of multiple inhouse systems, each associated with a single network protocol, these systems can send PC/PHCC data to a destination host outside of the enterprise network through the active warden without causing protocol switches. Therefore, a coordination of the distributed sender hosts is necessary but can be achieved by introducing a single coordinator host connected to the actual PC/PHCC senders. However, due to the presence of the active warden, the covert channel sender is forced to coordinate its distributed transfer and the command and control traffic between the covert channel sender hosts and the covert channel coordinator system are required to be hidden as well, i.e., will raise the chance of a detection.

A similar approach is a **Covert Channels with Multiple Receivers:** If one covert channel host sends packets to different covert channel receivers and each covert channel receiver is associated with only a single protocol, no protocol switches between a single sender and a single destination occur (since each path is only used for one protocol) and no bitrate is limited in a direct way but in an indirect way: If the covert channel receiver is forced to be a distributed system (i.e., a covert channel-based botnet/zombie network), it has to implement a distributed coordination mechanism (sorting packets and extracting all hidden information on a single system that finally computes the whole hidden message). If the receiver-side network is monitored as well, the coordination itself must be covered too, and thus can probably be detected or at least raise attention. Also, the bitrate is limited since multiple receivers can receive messages via network access points with different performance and the network packets for the coordination can differ in their timings, too. Therefore, the sender must introduce pause intervals (which limit the bitrate) between new packets to prevent a jumbled result at the receiver.

Improved Control Protocols: PHCCs with internal control protocols can – if a sequence number is present – resort jumbled packet sequences at the receiver side and thus, can make the active warden inefficient even if delays are introduced. However, applied delays can cause the active warden to report abnormal protocol switching traffic and it would be useful, if the size of PHCC-internal control protocols could can be shrinked to prevent such attention raising behavior.

Therefore, we developed a technique called *status updates* [48]. The concept of status updates is to only transfer a header component to the receiver, if the last *status* of the header component changes. For instance, imagine a PHCC between a system A and C via a proxy B, i.e., $A \rightarrow B \rightarrow C$. Therefore, A configures the covert channel proxy B to forward data to C via sending a status update that defines the forwarding destination C. All following packets received by B from A will be forwarded to C and A is not required to define the covert payload's destination C again. However, A can send a new status update for the destination to B to, for instance, send all following traffic to D (instead of sending it to the previously configured destination C). Status updates work for all status-based header components (e.g., also for source addresses or fields to indicate the start/end of a transaction) and can therefore decrease header sizes.

We combined our status update approach in [48] with dynamic routing for covert channel overlays. If a PHCC sender can measure a delay applied on protocol switches between itself and the PHCC receiver by ping'ing the receiving peer with and without causing a protocol switch, it can determine the possible presence of an active warden. In such a case, the PHCC sender could try to find an alternative routing path that does not face delays on protocol switches. Afterwards, the PHCC can even be transformed into a PC since no delay is applied and no packet jumbling will occur. To achieve this goal, we developed the *Smart Covert Channel Tool* – a dynamic routing middleware capable of utilizing various covert channel technologies, which can include PCs, but other covert channel techniques, such as network timing channels and network storage channels, as well.

VI. Improvement Proposal 1: Applying Formal Grammar to Increase Practical Usage

As mentioned earlier, our active warden must comprise an additional means to handle selected practical issues. We therefore propose a means based on formal grammar that addresses the following previously discussed problems:

1) The problem of delaying legitime protocol switches (e.g., DNS $\rightarrow$ HTTP),
2) the problem of host-related protocol switches (e.g., a client downloads from a web-server while sending a large e-mail to the SMTP server),
3) the problem of running multiple services on a single host (e.g., SMTP and IMAP service on the same machine).

Formal grammar has previously been applied in the context of computer security. For instance, Gorodetski et al. used formal grammar for attack modelling in [49] and Trinius and Freiling realized SPAM filters with context-free grammars [50].

A formal grammar $G = (V, \Sigma, P, S)$ comprises the set V of non-terminals, the set Σ of terminal symbols, the set P of productions and the starting symbol $S \in V$ [49].

Our formal grammar-based approach is based on the idea of *whitelisting*, i.e., we define the allowed protocol switching behavior within in the context of a company's network within our formal grammar and afterwards test, whether the network's actual behavior is conform to the rules of the grammar. If the protocol switching behavior *is* conform, the active warden does *not* apply a delay, otherwise it will delay packets.

Therefore, we first define the allowed network protocols as terminal symbols p_x where x is the protocol, e.g., $\Sigma_1 = \{p_{dns}, p_{http}, p_{https}, p_{smtp}, p_{imap}\}$. Afterwards, we define additional terminal symbols for the servers s_y in our network where y is the address of the server (it can be an IP or a hostname, e.g., $\Sigma_2 = \{s_{mail}, s_{name}, s_{web}\}$). Both sets form the set of terminal symbols $\Sigma = \Sigma_1 \cup \Sigma_2$.

In the next step, we define the production rules in P. We define productions, which are built in the form *<server>* *<protocol>*, e.g., $s_{mail}p_{smtp}$, which can comprise allowed rules of the utilized grammar type (e.g., context-free or regular).

The following example grammar allows

1) the use of SMTP and HTTP (both also after DNS), and
2) to switch from HTTP to HTTPS and vice versa, i.e., we support HTTP-based websites with HTTPS links/content (and vice versa),
3) as well as to switch between SMTP and IMAP to allow sending and receiving e-mails at the same time;
4) to run different services on the same machine (here, the SMTP and IMAP services are located on the same machine s_{mail});
5) the simultanous use of e-mail, DNS and web access, since W_2 allows the production M_2 and since M_2 allows the production W_2.

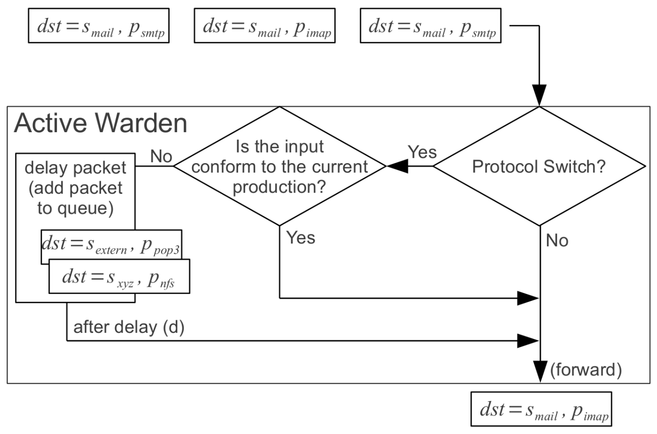

Figure 8. The integration of grammar productions and terminal symbols in the active warden.

$$
\begin{aligned}
V &= \{S, D, W_1, W_2, M_1, M_2\} && (3)\\
P &= \{S \to D(W_1|M_1)|W_1|M_1 && (4)\\
&\quad D \to s_{name}p_{dns} && (5)\\
&\quad W_1 \to s_{web}(p_{http}|p_{https})W_2 && (6)\\
&\quad W_2 \to DW_1|W_1|M_2|\epsilon && (7)\\
&\quad M_1 \to s_{mail}(p_{smtp}|p_{imap})M_2 && (8)\\
&\quad M_2 \to DM_1|M_1|W_2|\epsilon\} && (9)
\end{aligned}
$$

If a new packet arrives at the active warden, the active warden tries to find a production rule that fits the protocol sequence – either by starting a production from scratch or by continuing an existing one – in other words, the active warden tries to find a suitable production to allow a direct forwarding. Figure 8 visualizes this concept.

For instance, if a DNS packet was received (e.g., identified by the DNS port or by advanced protocol identifying tests like [51]), it is not delayed because a production rule ($S \to D(W_1|M_1)$) allows the DNS packet. Afterwards, an SMTP packet arrives, which is also allowed by the production rules ($S \to D \to M_1$).

If multiple packets of the same protocol (e.g., many SMTP packets for sending a larger e-mail or multiple HTTP packets from a file download) occur, they are not delayed because no protocol switch is taking place. Thus, non-protocol switching rules are not required, which ensures small grammar productions.

As an additional means to keep grammars as small as possible, we propose to implement layer-based grammars for the application of rules specific to the network layer of the TCP/IP model. Low-level protocol switching covert channels are not considered a threat in that case because they are not available for routing and low-level frames will not directly pass the active warden. For instance, an ARP request need not be modelled in the grammar nor must it be handled by the active warden since the active warden will not forward ARP requests (as long as it does not explicitly act as an ARP proxy).

VII. Improvement Proposal 2: Detection-capable Active Warden to Counter PCs

In recent work [35], we evaluated the detectability of protocol channels. We presented two algorithms for a PC detection. The first algorithm uses a static formula for a traffic detection, the other algorithm uses machine learning based on the C4.5 algorithm to build a decision tree and provides better results than the first algorithm and shall be discussed in this Section.

The C4.5 algorithm requires a traffic recording of at least a few thousand packets (in our tests, we used 5000 packets) and thus, only existing protocol channels that already transferred information can be detected and the continuing covert channel transmission can be interrupted (e.g., blocked by a firewall or delayed by the active warden).

For the detection of protocol channels, we use the change rate R that reflects how frequently protocol switches for a given sender occur (C is the total number of protocol switches and P the total number of packets of a traffic recording).

$$R = \frac{C}{P} \qquad (10)$$

A second parameter introduced is the average time between protocol switches D (T_i is the time of a protocol switch i).

$$D = \frac{\sum_i (T_{i+1} - T_i)}{C} \qquad (11)$$

We observed, that protocol channels are linked to higher R and/or smaller D values than normal traffic [35].

Since our technique provides an accuracy of 98-99% for PCs with a bitrate of $4\ bits/s$ or higher, the decision tree-approach can be considered useful in practice if used in conjunction with the active warden. Therefore, the active warden would act as described but would stop applying a delay (or would decrease the applied delay) on traffic that comprises no PC.

As mentioned, the machine learning-based detection approach requires a traffic recording. The active warden must therefore record the traffic on the fly and would only apply delays if enough packets were recorded and if the traffic got classified as being covert channel traffic (optimal results require $n = 5000$ packets). Thus, the active-warden's detection capability would not be usable for the first n packets.

Our problem of requiring enough traffic data before being able to act as required is similar to another kind of active warden: the traffic normalizer. Traffic normalizers face a so-called *cold start problem* [7]: If a traffic normalizer bootstraps, it receives packets of already existing connections and does not know any details of previously sent packets (e.g., whether a received packet belongs to an actually established connection or not) and cannot apply all normalization rules without causing negative effects on the traffic.

However, after a first traffic recording of n packets got classified, the active warden must continue recording traffic in a queue. The queue should contain $n = 5000$ packets at all times to provide optimal detection results. Therefore, the oldest packet is removed when a new packet is added to the queue. A queue also prevents continuously growing buffers that decrease the active warden's performance over the time.

Since a traffic classification is time-consuming and the performance of the active warden must be ensured, it is not recommendable to classify the traffic recording for each received packet. Instead, the number of packets $< n$ before a new C4.5 classification is applied should be adjusted to the throughput of the links (e.g., every 1000 or 2500 packets).

The active warden can either stop applying delays on traffic if a traffic was classified as *not* being a covert channel, or it can decrease the applied delay. On the other hand, it can apply the delay (or increase the delay) if a traffic was classified as being covert channel traffic.

VIII. Conclusion and Future Work

In this paper, we present the first active warden designed to counter both types of protocol switching covert channels: protocol channels (PCs) as well as protocol hopping covert channels (PHCCs). We limit the useful bitrate of these covert channels by disturbing the protocol switches through synthetically introduced delays. We implemented an active warden with both constant and randomized delay, based on *netfilter/iptables*. The active warden is suitable for a practical use but can still result in side-effects on regular traffic. Therefore, we proposed to combine the active warden with a detection functionality for protocol channels and additionally, to apply whitelisting based on a formal grammar to prevent that delays are applied to legitimate network traffic.

Future work will include to find solutions for the problem of network address translation (NAT) inside a protected network as well as to find solutions for effects of large network environments where load balancing and redundancy protocols are required; the presented prototype was not designed for such environments. Additionally, research must be done to provide an exact bitrate controlling for PHCCs using internal sequence numbers since these channels can re-sort jumbled packet sequences caused by the active warden. We do not expect our approach to be easily modifiable to counter such advanced protocol hopping covert channels.

Acknowledgement

We would like to thank Sebastian Zander for his comments on the calculation of the max. error-free bitrate.

References

[1] S. Wendzel and J. Keller, "Design and implementation of an active warden addressing protocol switching covert channels," in *Proc. 7th International Conference on Internet Monitoring and Protection (ICIMP 2012)*, A. Wagner and P. Dini, Eds. IARIA, 2012, pp. 1–6.

[2] B. W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[3] S. J. Murdoch, "Covert channel vulnerabilities in anonymity systems," Ph.D. dissertation, University of Cambridge, 2007.

[4] *Trusted Computer System Evaluation Criteria (TCSEC, Orange Book)*, Department of Defense (DoD) Std., Aug 1985.

[5] J. P. R. Gallagher, Ed., *A Guide to Understanding Covert Channel Analysis of Trusted Systems (NCSC-TG-030)*. National Computer Security Center, Nov 1993.

[6] S. Zander, G. Armitage, and P. Branch, "Covert channels and countermeasures in computer network protocols," *IEEE Comm. Magazine*, vol. 45, no. 12, pp. 136–142, Dec 2007.

[7] M. Handley, V. Paxson, and C. Kreibich, "Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics," in *Proc. 10th USENIX Security Symposium*, 2001, pp. 115–131.

[8] A. Singh, O. Nordström, A. L. M. dos Santos, and C. Lu, "Stateless model for the prevention of malicious communication channels," *Int. Journal of Comp. and Applications*, vol. 28, no. 3, pp. 285–297, 2006.

[9] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proc. USENIX Security Symp.*, 2008, pp. 139–154.

[10] Daemon9, "Loki2 (the implementation)," *Phrack Magazine*, vol. 7, no. 5, September 1997, retrieved: Dec, 2012. [Online]. Available: http://www.phrack.org/issues.html?issue=51&id=6

[11] S. Wendzel and J. Keller, "Low-attention forwarding for mobile network covert channels," in *Proc. 12th IFIP Conf. on Communications and Multimedia Security*. Springer LNCS vol. 7025, 2011, pp. 122–133.

[12] S. Wendzel, "Protocol hopping covert channels," *Hakin9*, vol. 08, no. 03, pp. 20–21, 2008, (in German).

[13] ——, "Protocol channels as a new design alternative of covert channels," *CoRR*, vol. abs/0809.1949, pp. 1–2, 2008.

[14] C. G. Girling, "Covert channels in LAN's," *IEEE Transactions on Software Engineering*, vol. 13, pp. 292–296, February 1987.

[15] M. Wolf, "Covert channels in LAN protocols," in *Proc. Local Area Network Security*. Springer LNCS vol. 396, 1989, pp. 89–101.

[16] C. H. Rowland, "Covert channels in the TCP/IP protocol suite," *First Monday*, vol. 2, no. 5, May 1997, retrieved: Dec, 2012. [Online]. Available: first-monday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/528/449

[17] J. Rutkowska, "The implementation of passive covert channels in the linux kernel," 2004, ftp://ftp.pastoutafait.org/pdf/passive-covert-channels-linux.pdf, retrieved: Dec, 2012.

[18] S. Cabuk, C. E. Brodley, and C. Shields, "IP covert timing channels: design and detection," in *Proc. ACM Conference on Computer and Communications Security*, 2004, pp. 178–187.

[19] K. Ahsan and D. Kundur, "Practical data hiding in TCP/IP," in *Proc. Workshop on Multimedia Security at ACM Multimedia '02*, French Riviera, December 2002.

[20] R. Rios, J. Onieva, and J. Lopez, "HIDE_DHCP: Covert communications through network configuration messages," in *Proc. IFIP TC 11 27th International Information Security Conference, Heraklion, Crete, Greece*. Springer, 2012.

[21] N. Lucena, G. Lewandowski, and S. Chapin, "Covert channels in IPv6," in *Proc. Privacy Enhancing Technologies*. Springer LNCS vol. 3856, 2006, pp. 147–166.

[22] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts, "Covert messaging through TCP timestamps," in *Proc. 2nd international conference on privacy enhancing technologies*. Springer, 2003, pp. 194–208.

[23] S. J. Murdoch and S. Lewis, "Embedding covert channels into TCP/IP," in *Proc. Information Hiding Conference 2005*. Springer LNCS vol. 3727, 2005, pp. 247–261.

[24] R. Accorsi and C. Wonnemann, "Detective information flow analysis for business processes," in *Proc. Business Process, Services Computing and Intelligent Service Management*. GI LNI vol. 147, 2009, pp. 223–224.

[25] M. H. Kang, I. S. Moskowitz, and S. Chincheck, "The pump: A decade of covert fun," in *Proc. 21st Annual Computer Security Applications Conference (ACSAC 2005)*. IEEE Computer Society, 2005, pp. 352–360.

[26] N. Ogurtsov, H. Orman, R. Schroeppel, S. O'Malley, and O. Spatscheck, "Covert channel elimination protocols," University of Arizona, Tech. Rep., 1996.

[27] P. A. Porras and R. A. Kemmerer, "Covert flow trees: A technique for identifying and analyzing covert storage channels," in *Proc. IEEE Symp. on Security and Privacy*, 1991, pp. 36–51.

[28] R. A. Kemmerer, "Shared resource matrix methodology: an approach to identifying storage and timing channels," *ACM Trans. Comput. Syst.*, vol. 1, no. 3, pp. 256–277, 1983.

[29] J. McHugh, "An information flow tool for gypsy - an extended abstract revisited," in *Proc. 17th Annual Computer Security Applications Conference*, 2001, pp. 191–201.

[30] J. Agat, "Transforming out timing leaks," in *Proc. 27th ACM Symp. on Principles of Programming Languages (POPL)*. ACM Press, 2000, pp. 40–53.

[31] C. Krätzer and J. Dittmann, "Früherkennung von verdeckten Kanälen in VoIP-Kommunikation," in *IT-Frühwarnsysteme (BSI-Workshop)*. BSI, 2006, pp. 209–214, (In German).

[32] W.-M. Hu, "Reducing timing channels with fuzzy time," in *Proc. 1991 Symposium on Security and Privacy, IEEE Computer Society*, 1991, pp. 8–20.

[33] V. Berk, A. Giani, and G. Cybenko, "Detection of covert channel encoding in network packet delays," Department of Computer Science - Dartmouth College, Tech. Rep., 2005.

[34] R. deGraaf, J. Aycock, and M. J. Jacobson, "Improved port knocking with strong authentication," in *Proc. 21st Annual Computer Security Applications Conference (ACSAC '05)*. IEEE Computer Society, 2005, pp. 451–462.

[35] S. Wendzel and S. Zander, "Detecting protocol switching covert channels," in *Proc. 37th IEEE Conf. on Local Computer Networks (LCN)*. IEEE, 2012, pp. 280–283.

[36] F. V. Yarochkin, S.-Y. Dai, C.-H. Lin, Y. Huang, and S.-Y. Kuo, "Towards adaptive covert communication system," in *Proc. 14th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2008, pp. 153–159.

[37] W. Li and G. He, "Towards a protocol for autonomic covert communication," in *Proc. 8th Conf. on Autonomic and Trusted Computing*, 2011, pp. 106–117.

[38] C.-R. Tsai and V. D. Gligor, "A bandwidth computation model for covert storage channels and its applications," in *Proc. IEEE Conf. on Security and Privacy*, 1988, pp. 108–121.

[39] C. D. Mee and E. D. Daniel, *Magnetic Storage Handbook*, 2nd ed. McGraw Hill, 1996.

[40] B. Ray and S. Mishra, "A protocol for building secure and reliable covert channel," in *Proc. 6th Annual Conference on Privacy, Security and Trust (PST 2008)*. IEEE, 2008, pp. 246–253.

[41] D. Stødle, "Ping tunnel – for those times when everything else is blocked," 2009. [Online]. Available: http://www.cs.uit.no/˜daniels/PingTunnel/

[42] S. Wendzel, "pct," 2009, retrieved: Dec, 2012. [Online]. Available: http://www.wendzel.de/dr.org/files/Projects/pct/

[43] D. Berrange, "Simulating WAN network delay," 2005, retrieved: Dec, 2012. [Online]. Available: http://people.redhat.com/berrange/notes/network-delay.html

[44] J. Morris, "IPTables::IPv4::IPQueue module for Perl," 2002, retrieved: Dec, 2012. [Online]. Available: http://search.cpan.org/˜jmorris/perlipq-1.25/IPQueue.pm

[45] S. Wendzel, "phcct," 2007, retrieved: Dec, 2012. [Online]. Available: http://www.wendzel.de/dr.org/files/Projects/phcct/

[46] T. Kohno, A. Broido, and K. Claffy, “Remote physical device fingerprinting,” *IEEE Transactions on Dependable and Secure Computing*, no. 2, pp. 93–108, 2005.

[47] Akamai, “Retail web site performance,” 2006, retrieved: Dec, 2012. [Online]. Available: http://www.-akamai.com/dl/reports/Site_Abandonment_Final_Report.pdf

[48] P. Backs, S. Wendzel, and J. Keller, “Dynamic routing in covert channel overlays based on control protocols,” in *Proc. International Workshop on Information Security, Theory and Practice (ISTP-2012)*. IEEE, 2012, pp. 32–39.

[49] V. Gorodetski and I. Kotenko, “Attacks against computer network: Formal grammar-based framework and simulation tool,” in *Proc. Recent Advances in Intrusion Detection*. Springer LNCS vol. 2516, 2002, pp. 219–238.

[50] P. Trinius and F. Freiling, “Filtern von Spam-Nachrichten mit kontextfreien Grammatiken,” in *Proc. Sicherheit 2012*. GI LNI vol. P-195, 2012, pp. 163–174, (in German).

[51] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, “Traffic classification on the fly,” *SIGCOMM Computer Communication Review*, vol. 36, no. 2, pp. 23–26, Apr. 2006.