The Silence of the LANs: Efficient Leakage Resilience for IPsec VPNs

Steffen Schulz
Ruhr-University Bochum &
Macquarie University
Bochum, Germany
steffen.schulz@rub.de

Ahmad-Reza Sadeghi
TU Darmstadt (CASED) &
Fraunhofer SIT
Darmstadt, Germany
ahmad.sadeghi@cased.de

Vijay Varadharajan
Macquarie University
Sydney, Australia
vijay@science.mq.edu.au

ABSTRACT

Virtual Private Networks (VPNs) are increasingly used to build logically isolated networks. However, existing VPN designs and deployments neglected the problem of traffic analysis and covert channels. Hence, there are many ways to infer information from VPN traffic without decrypting it. Many proposals have been made to mitigate network covert channels, but previous works remained largely theoretical or resulted in prohibitively high padding overhead and performance penalties.

In this work, we (1) analyse the impact of covert channels in IPsec, (2) present several improved and novel approaches for covert channel mitigation in IPsec, (3) propose and implement a system for dynamic performance trade-offs, and (4) implement our design in the Linux IPsec stack and evaluate its performance for different types of traffic and mitigation policies. At only 24% overhead, our prototype enforces tight information-theoretic bounds on all information leakage. To encourage further research on practical systems, our prototype is available for public use.

Keywords

IPsec, VPNs, covert channels, performance trade-offs

1. INTRODUCTION

Virtual Private Networks (VPNs) are popular means for enterprises and organizations to securely connect their network sites over the Internet. Their security is implemented and enforced by VPN gateways that tunnel the transferred data in secure channels, thus logically connecting the remote sites in an isolated network. Abstracted this way, VPNs are increasingly used in scenarios that secure channels were not designed for: to logically isolate networks, providing "networks as a service" in virtualized environments like Clouds, Trusted Virtual Domains, or the Future Internet [8–10]. What is not considered in these scenarios is the long known problem of covert channels.

Covert channels violate the system security policy by using channels "*not intended for information transfer at all*" [25, 35]. While there is a large body of research on covert channels, few works have considered the practical implementation and performance impact of comprehensive covert channel mitigation in modern networks. We believe such work is important for a number of reasons, especially regarding VPNs and network virtualization:

(1) Insider Threat: In contrast to end-to-end secure channels, where the endpoints are implicitly trusted, VPNs are also used for logical network isolation and perimeter security enforcement. In this context, the members of a VPN are often not fully trusted, but instead the trust is reduced to central policy enforcement points, the VPN gateways, which should prevent undesired information flows. However, malicious insiders in the LAN may leak information through the VPN gateways using covert channels, thus circumventing the security policy. Examples of such insiders can be actual humans or stealth malware, engaging in industrial espionage, leaking realtime financial transaction data, or disclosing large amounts of data from physically secured institutions (e.g., to Wikileaks).

(2) Traffic Analysis: By analysing traffic patterns and meta-data, it is also possible to infer information about transferred data without assuming a malicious insider [26, 44]. Such "passive" Man-in-the-Middle (MITM) scenarios are becoming more prevalent with network virtualization, allowing co-located, supposedly isolated systems to analyse each other [38]. To mitigate such attacks, a common approach is to consider the maximum possible information leakage by a colluding malicious insider. In limiting this maximum information leakage, covert channel analysis and mitigation thus also affects traffic analysis [19].

(3) Combination with Detection: Although application-layer firewalls and intrusion detection systems are widely deployed, carefully designed covert channels remain hard to detect [27, 34]. In these systems, the adversary chooses a weaker signal and mimics the patterns of regular channel usage. Covert channel mitigation can be useful here to induce noise, forcing the adversary to use a stronger signal and thus facilitate detection. We expect the combination of covert channel mitigation and detection to significantly reduce the performance penalty of covert channel mitigation by allowing less intrusive pattern enforcement.

Contributions.

This paper provides for the first time an explicit analysis

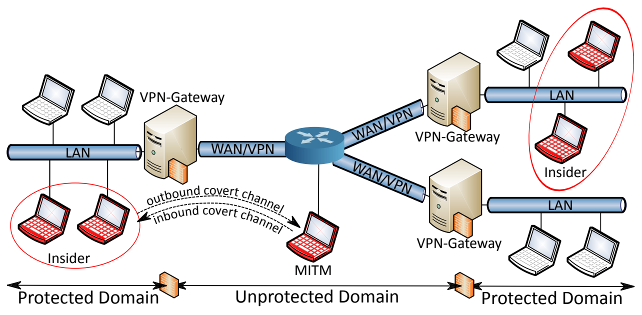

Figure 1: Problem scenario: A VPN with three LAN sites. The adversary aims to exchange information between the MITM and malicious insiders using covert channels.

of covert channels in IPSec based VPNs and a comprehensive set of techniques and mechanisms to mitigate them. We identify and categorize the different types of covert channels and determine their capacity. We develop a framework for mitigation of these covert channels and describe mechanisms and techniques for high-performance covert channel mitigation. In particular, we propose an algorithm for on-demand adjustment of traffic pattern enforcement that increases peak network performance while also reducing overhead during reduced usage. We present a practical instantiation of this framework for the Linux IPSec stack and analyse its performance for different kinds of traffic. In contrast to previous works, which achieve throughput rates in the range of modem speed [19, 42] and taunt the performance impact of proposed mitigation mechanisms [31], our prototype achieves 169 Mbit/s in a 200 Mbit/s VPN link at only 24% overhead.

Outline.

After defining the problem of VPN covert channels in Section 2, we discuss efficient covert channel mitigation and performance trade-offs in Section 3. An implementation for the Linux IPsec stack is presented and evaluated in Section 4. We discuss related work in Section 5 and conclude in Section 6. A detailed discussion of the available covert channels in IPsec is provided in Appendix A.

2. PROBLEM SETTING AND ADVERSARY MODEL

In the following we define the problem of covert channels in VPNs. Note that our definition differs from previous, less explicit considerations, which consider communication between legitimate VPN participants and are better described as steganographic channels [2,22,24]. Although we limit ourselves to VPNs in state-of-the-art IPsec configuration [11], most of our results can be generalized.

2.1 System Model and Terminology

As illustrated in Figure 1, we consider a VPN comprised of two or more Local Area Networks (LANs) that are interconnected over an insecure Wide Area Network (WAN). In our scenario, the security goal of the VPN is not only to provide a secure channel (confidentiality, authenticity, integrity) but also to *confine* communication of LAN hosts to the VPN, i.e., to isolate the *protected* from the *unprotected* domain. VPNs are increasingly used for such logical isolation, to create secure virtualized or overlay networks, or simply enforce perimeter security in large companies [8–10]. This de-facto security goal of isolating the protected from the unprotected domain, and its efficient implementation, is the main focus of this work.

For this purpose, we distinguish *legitimate* channels that transfer and protect user data according to the VPN security policy from *covert* channels that can be used to circumvent this policy. Covert channels exist because the legitimate channel acts as a shared resource between the protected and unprotected domain, exhibiting certain *characteristics* that can be manipulated and measured by different parties. We denote channels from the protected to unprotected domain and vice versa as *outbound* and *inbound* covert channels, respectively.

We measure the security of our system using the Shannon capacity of the covert channels, i.e., the information theoretic limit on the amount of information that can be transferred through them [44]. The covert channel capacity is given in bits per legitimate channel packet (bpp) or, where applicable, in bits per second (bps). The capacity of each covert channel *type* is denoted as C^{type}. The capacities are classified as maximum (m) vs. remaining (r) covert channel rate for inbound (in) vs. outbound (out) covert channels. For example, the maximum capacity of the outbound covert channel based on packet size is denoted as $C^{\text{PktSize}}_{m,out}$, or as $C^{\text{PktSize}}_{r,out}$ after countermeasures have been applied. The remaining *aggregated* inbound and outbound covert channel rates are denoted as $\hat{C}_{r,in}$ and $\hat{C}_{r,out}$, respectively.

2.2 Adversary Model

The adversary controls one or more compromised hosts in the LAN sites as well as an active MITM in the WAN. We refer to the LAN hosts controlled by the adversary as (malicious) *insiders*, regardless of whether they are controlled by actual humans or stealth malware. The adversary's goal is to establish a communication channel between the MITM and one or more possibly colluding malicious insiders, as illustrated in Figure 1. This would allow the adversary to send instructions to the insiders or to leak information from the protected to the unprotected domain, breaching the perimeter security of the VPN. For this purpose, we assume a state-of-the-art IPsec configuration with authenticated encryption using ESP in tunnel mode [11], and the cryptographic primitives and keys of the VPN are securely enforced by the VPN gateways. However, the legitimate VPN traffic can be manipulated by malicious parties in the protected and unprotected domains to exchange information that "survives" these packet transformation enforced by the VPN gateways.

Unfortunately, no systematic approach is known for identifying network covert channels apart from exhaustive search, and the categorization as storage or timing channels can be ambiguous [35]. We used a comprehensive analysis on the IPsec specification and related work on covert channels in network protocols (cf. Section 5), as well as source code analysis and testing[1] to identify potential covert channels in IPsec VPNs. IP-Tunneling and authenticated encryption by the IPsec gateways greatly simplified this problem, as none of the protocol headers that the MITM can read or

[1]Specifically, we examined the IPsec implementations of the current Linux 2.6.32 to 2.6.38 and OpenBSD 4.7 to 4.8 releases.

Class	Type	Capacity C_m in bpp	
		Outbound	Inbound
storage	ECN	2	1
	DS	6	6
	Flags	1	-
timing/ channel- logic	PktSize	8.4	-
	IPD	≥ 1	≥ 1
	PktOrd	-	> 6.58
	PktDrop	-	1
	PMTUD	-	5.18
amplify	DestIP	$\log_2(N)$	-

Table 1: Inbound and outbound covert channels capacities for an IPsec VPN with $N + 1$ endpoints.

modify (i.e., the outer IP and Encapsulated Security Payload (ESP) header) are directly available to the LAN hosts.

In total, we have identified only eight covert channels. As shown in Table 1, the available covert channels comprise three storage-based channels based on fields in the outer IP header (ECN, DS, Flags) and five timing-based covert channels that manipulate Inter-Packet Delay (IPD), packet order (PktOrd), WAN capacity (PktDrop), and Path MTU Discovery (PMTUD). The remaining characteristic of the respective destination LAN of a packet (DestIP) does not constitute a covert channel in its own right but can act as amplification of other covert channels. A detailed discussion of the covert channels we identified in IPsec VPNs is available in the full version [39].

We emphasize that some of these channels are implementation dependant, e.g., the treatment of ECN header flags or PMTUD at the VPN gateway, while others (IPD, PktSize, PktOrd) are generic problems faced by all packet-oriented channels. While we are confident to have identified all covert channels, we cannot account for all possible implementations and interpretations of IPsec. Hence in this paper we limit our considerations to the identified attack vectors.

3. COVERT CHANNEL-RESILIENT IPSEC

In this section we present the design of a high-performance covert channel-resilient IPsec, i.e., a system with low, known covert channel capacity and high throughput. We present novel or improved techniques for efficient covert channel mitigation in Section 3.1. Section 3.2 considers the performance of different mitigation strategies, introducing on-demand performance trade-offs. Finally, we derive the remaining aggregated inbound and outbound covert channel capacities of the system in Section 3.3.

3.1 Covert Channel Mitigation

In the following we present and improve efficient mitigation mechanisms for each of the covert channels identified in Section 2.2.

3.1.1 Packet Size (PktSize)

The packet size characteristic is usually addressed by padding packets to maximum size or assuming them to be of constant size [44]. However, as the product throughput = pkt_size · pkt_rate is constant for a given link, enforcement of small packet sizes can reduce the load per packet significantly, allowing higher packet rates and more simultaneous connections.

It was previously proposed to allow multiple alternate packet sizes [18], but then the ratio between packets of different sizes creates another covert channel. Mode Security [7] was proposed to manage the switching between different enforcement modes and audit such a remaining covert channel. However, real network traffic is often mixed, i.e., packet streams using different packet sizes are often transmitted at the same time. Moreover, the enforcement of small packet sizes is problematic for IP protocols: With Path MTU Discovery (PMTUD), the connection endpoints quickly detect and adapt to the maximum allowed packet size of an IP route, but only slowly recover to a larger MTU using a conservative trial-and-error approach. This active adaption also makes it harder for the VPN gateways to estimate the actual demand for larger packets.

We address these problems by combining packet padding with transparent fragmentation and multiplexing, mechanisms that were previously only considered for traffic pattern obfuscation [23,45]. Packet fragmentation within IPsec allows us to efficiently and transparently enforce various packet sizes at the gateway without influencing the channel's Path MTU (PMTU). This is different from regular IP fragmentation before or after IPsec processing, which results in visible fragments either on the LAN or WAN sides that could again be used as covert channels. The fragmentation mechanism is complemented by packet multiplexing, which can be used to reduce packet padding overhead by concatenating multiple smaller packets up to the desired packet size. This also reduces the IPsec encapsulation overhead (ESP, IP).

When working with mixed traffic, the sender gateway first fragments large packets and then attempts to multiplex small packets or fragments into the padding area of previously processed packets that are still in the packet buffer. At the receiving gateway, packets are first de-multiplexed and then defragmented. As this mechanism work transparently for the LAN sender and receiver, the LAN gateways can precisely monitor the current demands of the adjacent LAN site to optimally adjust the enforced packet size.

3.1.2 Inter-Packet Delay (IPD)

The covert channel based on IPDs and its mitigation were subject of several previous works (e.g., [13, 27, 33, 43, 44]). In theory, it is easily eliminated by enforcing a fixed IPD at the VPN gateway, inserting dummy packets when no real packets are available [43]. However, due to the very high packet rates in modern networks, even short periods of non-optimal enforcement of IPDs (and thus packet rate) at the VPN gateway quickly result in packet loss due to packet buffer overflows or network congestion. This is particularly critical for Internet protocols, where packet loss triggers congestion avoidance, degrading overall throughput independently of the packet rate enforced by the VPN gateways. The effect can be partly mitigated with large packet buffers; however, large buffers can also create high packet delays, degrading network responsiveness [16]. Also, the optimal enforced packet rate can be very large in modern networks, creating a high computational overhead for the time-synchronous packet processing. For example, to saturate a 100 Mbit/s link with 200 byte packets, an average IPD of $\frac{500\ byte}{100 \cdot 10^6\ byte/s} = 2\mu s$ should be enforced. Finally, one must consider inaccuracies in the timing enforcement that appear at high system loads [13, 15]: Since high activity on the LAN interface can influence the system load of the gateway, a LAN host may induce inaccuracies in the IPD enforcement of the gateway that can again be measured by the Man-in-the-Middle (MITM), yielding $C_r^{\mathrm{IPD}'} = 0.16$ bps [19].

We have implemented the traffic reshaping inside the Linux kernel, using the modern High-Precision Event Timer (HPET) infrastructure for packet scheduling with nanosecond resolution. This substantially reduces the overhead of context switching and buffering, allowing an IPDs in the range of microseconds rather than several milliseconds (e.g., [19, 42]) and noticeably improves throughput and responsiveness. To maintain good system performance at even higher packet rates we use packet bursts, i.e., we translate very low IPDs into bursts of multiple packets at correspondingly larger delays. For optimal packet buffering our system adjusts the buffer size depending on the currently enforced IPD. This prevents long delays at low rates while allowing generous buffering at high rates.

To address the problem of timing inaccuracies, we use the high resolution of the HPET timers to monitor and actively *compensate* for timing inaccuracies in randomized IPD enforcement. Specifically, we exploit the fact that determining timing inaccuracies during randomized IPD enforcement is harder for the remote MITM than for the local system. The adversary always requires significantly more measurements to first detect the variance of the random IPD enforcement and then the inaccuracy in the enforced variance [13], while the VPN gateway itself can directly compare the intended versus actual packet sending time. Hence, the gateway can approximate the current inaccuracy faster, requiring less measurement samples. Given this knowledge of unintended change in IPD variance, we let the VPN gateways compensate for the enforcement inaccuracy by dynamically compensating the variance of the IPD enforcement. This prevents the adversary from ever measuring the actual inaccuracy, eliminating the timing channel ($C_r^{\mathrm{IPD}} = 0$). However, further evaluation with specialized network hardware is needed to confirm (the non-existance of) this effect.

3.1.3 Packet Order (PktOrd)

Sequence numbers in protocol headers have been used before to create a covert or steganographic channel based on packet reordering [12, 24]. However, in contrast to previous works we can eliminate this channel in the VPN scenario using the IPsec anti-replay window and secure sequence numbers in Encapsulated Security Payload (ESP).

IPsec implementations maintain a bitmap of the last r seen and unseen sequence numbers so that replay attacks within the window size can be detected and older packets discarded. To eliminate communication through packet reordering, we propose to implement this window as a packet buffer, where new packets are inserted *sorted* by their ESP sequence number and leave the buffer as the window advances. As a result, all packets forwarded from the VPN gateway into the LAN are ordered and the covert channel is eliminated: $C_{r,in}^{\mathrm{PktOrd}} = 0$.

Unfortunately, the approach is problematic for low packet rates, since the window may advance slowly and individual packets are not forwarded fast enough. We solve this issue by establishing a certain maximum IPD (e.g., 50ms) at the sender and assure that at least r dummy packets are sent by a gateway before a connection is stopped. These constraints are necessary in any case to assure network responsiveness and hide short periods of inactivity.

3.1.4 Packet Drops (PktDrop)

In general, it appears impossible to eliminate covert channels based on packet dropping in the WAN. Mitigation with error correction codes is expensive and easily defeated by dropping even more packets. Instead, we propose to mitigate the channel by injecting noise, by *increasing* packet loss proportionally to the actual packet loss.

Specifically, let the gateways maintain a buffer D of size l. At the sender gateway, packets are buffered in D and their order is randomized before encapsulation. At the receiver gateway, the packets are again collected in D and the number of dropped packets i in a sequence of l packets is determined based on their ESP sequence number. If $i > 0$, the gateway drops another j packets from the current buffer, such that $i + j = 2^x$, for $1 < x \leq \log_2(l)$, and forwards the remaining packets after randomizing their order once again. As a result, the MITM can choose the overall number of packets to be dropped for the receiving LAN client but cannot select which packets to drop, resulting in a symbol space of $\log_2(l) + 1$ packets per l packets. The remaining covert channel capacity is then $C_{r,in}^{\mathrm{PktDrop}} = \frac{1}{l} \cdot \log_2(\log_2(l)+1)$ bpp.

Similar to the packet re-ordering mitigation in Section 3.1.3, the inbound packet buffer D at the receiving gateway is problematic for very low traffic rates and requires similar restrictions to assure a steady stream of (dummy) packets.

Note that an optimized implementation could combine the packet sorting or IPD enforcement with the packet dropping facility to accumulation of delays through multiple queues.

3.1.5 Path MTU Discovery (PMTUD)

To our knowledge, no previous work considered the possibility of covert channels based on PMTUD, in particular with respect to VPNs. Since PMTUD is critical for good network performance, we do not disable it but instead mitigate the channel by enforcing limits on the rate and values that are propagated by the VPN gateways into the LAN.

In particular, we limit the possible PMTU values by maintaining a list of common PMTU values and only propagate the respective next lower PMTU to the LAN. Such common PMTUs values can be established on site or can be derived from previously proposed performance optimizations for PMTUD [32]. The rate limitation of PMTU propagation is problematic in general, as a lack of MTU adaption will lead to packet loss. However, in our case the current PMTU is always known to the trusted VPN gateways, which can then use the transparent fragmentation feature from PktSize enforcement to translate between LAN and WAN packet sizes. Considering the 10 most common PMTU values and an average interval of, e.g., 2 minutes [32] between propagation of PMTU changes, our measures reduce the covert channel rate to less than $C_{r,in}^{\mathrm{PMTUD}} = 0.03$ bps.

3.1.6 Storage-based Channels (ECN, DS, Flags)

The storage-based covert channels exploiting the Explicit Congestion Notification (ECN), Differentiated Services (DS) and IPv4 Flags handling of IP/IPsec are easily eliminated by resetting the respective fields of the outer IP header at encapsulation and ignoring them during decapsulation. Normalizing the IPv4 Flags field is unproblematic as en-route fragmentation is deprecated in IP. However, eliminating the ECN and DS covert channels disables these performance optimizations in the WAN.

3.2 Mitigation Policies and Performance

In this section, we discuss different covert channel mitigation policies that can be enforced using the techniques described in Section 3.1. We start by discussing the problems of previously proposed Fully Padded Channel and Mode Security approaches, and then propose a new system for on-demand, dynamic adaption of the enforced channel characteristics. We focus on the IPD and PktSize enforcement mechanisms, since they have by far the highest performance impact.

3.2.1 Fully Padded Channel

When applied without any performance trade-offs, the mitigation mechanisms described in Section 3.1 result in a *fully padded channel*: The WAN packet stream is constantly padded to the maximum desired throughput rate and packet size. However, this mitigation policy has several disadvantages: (1) The system must compromise between high throughput and responsiveness, likely opting to enforce maximum packet sizes to reduce fragmentation overhead; (2) the maximum (desired) network load is constantly enforced in both directions, reducing overall performance due to network congestion; (3) TCP/IP congestion avoidance algorithms do not work, since any rate throttling is compensated by additional channel padding. In case of temporary reductions in WAN capacity, this leads to repeated packet loss and throttling, until the network is not usable anymore. Hence, the fully padded channel policy is unfit for practical use, except in private/dedicated physical infrastructures.

3.2.2 Mode Security

Mode Security is a generic scheme for trading covert channel-resilience against system performance. This is done by organizing system operation in a set of alternative *operation modes* that can be switched at a certain rate [7]. The current operation mode is then selected such that performance penalty and/or overhead produced by the covert channel mitigation is minimized. Since the operation mode is typically adapted depending on the actually required usage, the adaption itself may be exploited as a covert channel. In this case, the covert channel capacity can be given as $C_{out}^{\text{ModeSec}} = R{\cdot}\log_2(M)$, where M is the number of operation modes and R is the maximum rate at which the operation mode can be changed (transition rate).

Mode Security was used to estimate the theoretic network overhead and covert channel capacity [44]. However, this assumes an algorithm that can determine the optimal operation mode to switch to. To the best of our knowledge, no practical implementation and evaluation of this mechanism exists; in particular, no strategies have been proposed to automatically determine and apply the optimal operation mode in the face of often unpredictable traffic, with exponential rate increases and congestion avoidance algorithms. In fact, our attempts to directly apply Mode Security to on-demand covert channel mitigation resulted in poor performance, with TCP throughput benchmarks becoming stuck at very low packet rates or completely losing the connection.

3.2.3 On-Demand Mode Security Management

An algorithm for on-demand adaptation in network covert channel mitigation must accommodate multiple conflicting constraints. It must quickly react to changes in channel usage to elude congestion avoidance algorithms, yet the amount of possible mode changes should be minimal. Moreover, the employed packet queue should buffer packet bursts at various average packet rates, yet react quickly when the current average rate is overused by dropping individual packets. We address these conflicts using the following regulation mechanisms:

Token Bucket Filter.

We generalize the transition rate R of the Mode Security paradigm to a token bucket filter [47]. Tokens are generated at a fixed rate R and each mode transition consumes a token from the token bucket. This allows us to "save up" unused mode transitions in form of tokens and consume them on demand, at temporarily higher rates than R. The amount of cached tokens is limited by the token bucket size and the *average transition rate* $\bar{R}$ is bound by the rate R at which new tokens are generated. Thus, the token bucket filter allows us to immediately react to changes in network usage, before connection throttling kicks in or network delays become noticeable. Further, the token bucket status may influence and optimize decisions on the operation mode to be enforced.

Aggressive Increase.

Network throughput is scaled mainly based on its packet rate r, with typically exponential rate increase until the first network bottleneck is detected. While the optimum WAN packet rate r_{opt} is easily calculated based on the currently observed LAN rate r_{LAN}, fragmented and multiplexed packets (r_{frag}, r_{mplex}), the derivation of the next enforced packet rate r_{new} is more involved, as shown in Algorithm 1.

Algorithm 1: Simplified pseudo-code for dynamic packet rate adjustment in steps of r_{quant}.

```
while *true* do
    $(r_{\text{LAN}}, r_{\text{frag}}, r_{\text{mplex}}) \leftarrow$ get-stats()
    $r_{\text{opt}} \leftarrow r_{\text{LAN}} + r_{\text{frag}} - r_{\text{mplex}}$
    $r_{\text{avg}} \leftarrow 0.1 \cdot r_{\text{opt}} + 0.9 \cdot r_{\text{avg}}$
    case $r_{\text{opt}} > 0.9 \cdot r_{\text{now}}$
        $r_{\text{amp}} \leftarrow (r_{\max} - r_{\text{opt}})/t_{\text{num}}$
        $r_{\text{new}} \leftarrow r_{\text{opt}} + \frac{1}{2} r_{\text{quant}} + r_{\text{amp}}$
    end
    case $r_{\text{now}} > 1.1 \cdot r_{\text{avg}} \wedge t_{\text{num}} > t_{\text{dec}}$
        $r_{\text{new}} \leftarrow r_{\text{avg}}$
    end
    $r_{\text{new}} \leftarrow$ quantatize($r_{\text{new}}, r_{\text{quant}}$)
    sleep($ival$)
end
```

To adequately consider exponential rate increases without requiring too frequent changes to r_{now}, our rate increase phase is designed to constantly *overestimate* the current optimal packet rate r_{opt}, by increasing r_{now} as soon as it is approached by r_{opt} (cf. Algorithm 1, Line 5). Combined with buffering and short monitoring intervals $ival \approx 200$ms, this approach successfully eludes congestion avoidance algorithms and prevents undesired throughput throttling. However, the overestimation should also not be too large, as it directly affects the padding overhead and can also reduce the inbound traffic rate due to the imposed network load.

Moreover, all stored tokens may be used up before a reasonably high packet rate $r_{\text{now}} \approx r_{\max}$ is reached, resulting in bad performance until new tokens are generated. Hence we also include an amplification mechanism that increases the rate r_{opt} in larger steps r_{amp}, depending on the currently available amount of tokens r_{num} (Line 6f.). This prevents the system from becoming "stuck" at low packet rates, at the cost of potentially high padding overhead in cases where such amplification was not required.

Conservative Slowdown.
When putting the WAN channel in a state of decreased performance, we must take care that sufficient transition tokens are available to adequately adapt to a possible subsequent usage increase as outlined above. In contrast to the aggressive rate increase policy, any reduction in the enforced traffic rate is therefore delayed until a certain amount of tokens t_{dec} have been collected in the token bucket. Moreover, to reduce the impact of short-term fluctuations in the packet rate, the rate is only reduced based on the longer-time average traffic rate r_{avg}, as shown in Algorithm 1 Line 8f. Overall, the described approach saves up tokens in the "slowdown" phases while aggressively spending them in the "increase" phase, creating an equilibrium around t_{dec} and r_{avg}.

Dynamic Queue Size with RED.
When dynamically adjusting the overall throughput of the WAN channel, we must also adjust the size of the packet queue accordingly. At small rates, a lot of packets may build up in a large queue, leading to large delays and timeouts. Similarly, a small queue is not effective at supporting a channel with high packet rates. Hence, we dynamically adapt the queue size based on the desired maximum buffering delay and the currently enforced packet rate. Eventually, the WAN channel or its enforcement policy may also reach a point where further rate increases are not possible. In this case, the endpoints should be notified of the current throughput limit as quickly as possible, without dropping several packets at once due to full buffers. We achieve this by deploying Random Early Detection (RED) [5] as the packet queue's dropping policy, so that packets are randomly dropped with increasing queue usage.

We implemented several variations of this approach and evaluated the effect of different parameters on the short-term and long-term usage adaption. The achieved performance and adaptation behavior is presented in Section 4.3.

3.3 Remaining Covert Channel Capacity

In the following we summarize the identified covert channels and derive the aggregated remaining covert channel capacity of our covert channel-resilient VPN.

Unfortunately, it is not possible to give all the covert channel rates in a closed form and with comparable units. Several covert channels also depend on additional parameters like network PMTU or minimum WAN packet rate. To provide a reasonable overview of the overall effectiveness of the covert channel mitigation, we have used the capacity estimations derived in the examples of Section 3.1, assuming a state-of-the-art IPsec VPN configuration (cf. Section 2).

Table 2 lists the individual covert channel capacities for the unmitigated (C_m) and mitigated (C_r) case. Considering that today's networks easily transmit several thousand

Type	**Max. Capacity** C_m **in bpp**		**Rem. Capacity** C_r **in bps**	
	Outbound	Inbound	Outbound	Inbound
ECN	2	1	0	0
DS	6	6	0	0
Flags	1	-	0	-
PktSize	8.4	-	0	-
IPD	≥ 1	≥ 1	0	0
PktOrd	-	> 6.58	-	0
PktDrop	-	1	-	≤ 5
PMTUD	-	5.18	-	0.03
DestIP	$\log_2(N)$	-	$\log_2(N)$	-
Overall	> 18.4	> 20.76	0	≤ 5.03

Table 2: Maximum and remaining covert channel capacities for VPNs with $N+1$ endpoints.

packets per second, i.e., 1 bpp $\gg$ 1 bps, our system results in significant improvements over standard IPsec. In fact, all outbound covert channels are completely eliminated, except for the DestIP channel. However, as explained in Section 2.2, the DestIP characteristic does not by itself constitute a covert channel but can only be used to amplify other channels. Hence, the overall remaining covert channel capacity is given by $\hat{C}_{r,out} = C_{out}^{\text{ModeSec}} \cdot C_{out}^{\text{DestIP}}$.

For the less critical inbound covert channels (e.g., control channels for stealth malware), only the channels based on PMTUD and PktDrop remain. The PktDrop covert channel has the highest impact with $C_{r,in}^{PktDrop} \leq 5$ bps and is easy to exploit. Since the PMTUD channel could be exploited at the same time, their capacities must be added up: $\hat{C}_{r,in} = C_{r,in}^{\text{PktDrop}} + C_{r,in}^{\text{PMTUD}} = 5.03$ bps.

4. PRACTICAL COVERT CHANNEL MITIGATION WITH LINUX

In this section we describe the instantiation of our system based on the Linux IPsec stack and analyse the achieved network performance and behavior.

In our prototype implementation and evaluation we only consider the mitigation of *outbound* covert channels, since information leakage from the protected to the unprotected domain is usually considered more critical (e.g., consider Bell-LaPadula [3]). Moreover, from our discussions in Section 3 it is clear that outbound covert channel mitigation is more efficient, as it requires less buffering and processing but is more effective in reducing the covert channel capacity.

4.1 Architecture and Implementation Details

We have implemented our design as an extension to the IPsec stack of the Linux kernel, called High-Performance Covert Channel Mitigation (HPCM). The implementation and is based on the Traffic Flow Confidentiality (TFC) project, a system for probabilistic traffic flow obfuscation and rerouting in IPsec [23]. We revised and extended TFC to support High-Precision Event Timers (HPETs), fragmentation, multiplexing, dummy packet generation that is indistinguishable from real traffic payloads, elimination of storage-based covert channels in the encapsulation headers and, most importantly, an interface for monitoring packet processing statistics and flexible configuration of the traffic pattern enforcement via userspace. The resulting architecture is il-

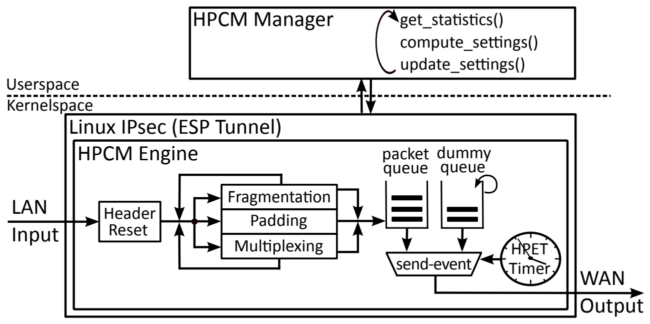

Figure 2: Architecture of our Linux prototype.

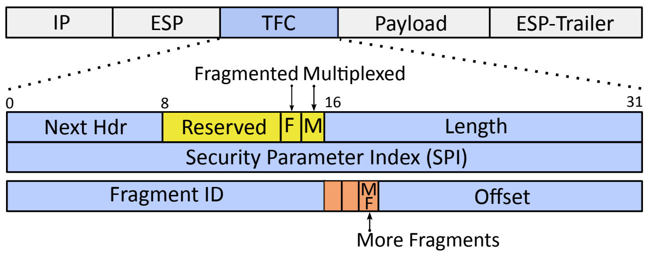

Figure 3: TFC encapsulation protocol.

lustrated in Figure 2. In kernelspace, the HPCM Engine processes packets as part of the IPsec subsystem, rewriting problematic header fields and enforcing the currently desired size and IPD constraints as described in Section 3.1. In userspace, the HPCM Manager collects processing statistics from the enforcement engine and combines them with the observed inbound LAN traffic to determine the optimal enforcement parameters, as presented in Section 3.2.3.

Packet processing.

Packet resizing is done on a best-effort basis to minimize processing delays. Packets that are too large are iteratively fragmented until the last remaining fragment is smaller or equal than the remaining packet size.

Packet multiplexing is performed based on two configurable thresholds, the flagging and the multiplexing threshold. Packets smaller than the flagging threshold are deemed to contain relatively large amounts of padding and are flagged as candidates for multiplexing before they are put into the outbound packet queue. Packets smaller than the multiplexing threshold are *first* considered for multiplexing, by searching the current packet queue for previously flagged packets with sufficient padding space and merging the current packet into such a previously processed packet. Only if no suitable candidate can be found in the outbound queue, packets smaller than the multiplexing threshold follow the regular size padding (and flagging) path.

This approach ensures that packet multiplexing has no adverse effect on the packet processing delays and avoids keeping extra state and timeouts for multiplexing candidates. In particular, the approach avoids cases where candidate packets for multiplexing are held up in a separate queue while the packet queue is empty, which would result in sending of dummy packets and negate any performance gained through packet multiplexing.

Since the resulting multiplexed packets typically contain a recent as well as an older packet that was temporarily exempted from the (concurrently running) packet sending process, they are inserted at the start of the packet queue to facilitate their fast sending.

We use the Linux sysfs filesystem to configure the kernelspace HPCM engine and report realtime processing statistics back to userspace. In particular, the HPCM engine exports counters for fragmented, padded and multiplexed packets as well as the amount of sent real and dummy packets. The counters can be periodically reset from userspace to yield average processing rates.

Protocol Format.

For flexible packet padding and rerouting, we deploy our own encapsulation protocol based on TFC [23] as shown in Figure 3. While the length field is sufficient to recognize and remove padding, we require two additional flags to mark packets as using fragmentation or multiplexing. TFC payloads are flagged as multiplexed if they are followed by another payload, and TFC payloads containing fragments are flagged as fragmented. Fragments payloads are accompanied with a 4 byte fragmentation extension header compatible with IPv4, which allows us to reuse the existing IP defragmentation support in the Linux kernel.

The resulting protocol overhead is relatively large, especially due to the additional Security Parameter Index (SPI) field which is used in IPsec to associate the stateless packet stream with some previously negotiated state at the VPN gateways, such as the encryption and authentication algorithms to be employed. However, in case of *outbound* traffic normalization such stateful processing at the receiver is not actually required. A more optimized implementation could integrate the encapsulation protocol into Encapsulated Security Payload (ESP), requiring only the two flags for marking fragmented and/or multiplexed packets and the optional 4 byte fragmentation extension header [2]

4.2 Testbed and Raw Performance

In this section we describe the performance achieved by our prototype in terms of network throughout, transaction rate (i.e., roundtrip time) and protocol overhead. Our testbed corresponds to the VPN scenario in Figure 1, except that we use only two LAN sites with one physical host per LAN. The Man-in-the-Middle (MITM) is implemented as an Ethernet bridge between the two VPN gateways, allowing reliable observation of all transmitted packets. For our evaluation, the MITM is completely passive and only used to provide independent performance measurements of the WAN. All hosts are 3.2 Ghz Intel Core i5-650 machines, equipped with two Intel PCIe GBit network cards and 4GB system memory. All network links are established at full-duplex GBit/s speed.

We have used the Netperf[3] benchmarks `TCP_STREAM` and `TCP_RR` to measure the maximum TCP throughput and transaction rate between the LAN sites. By comparing LAN and WAN throughput, we can determine the protocol overhead of the covert channel mitigation, including dummy packets and packet padding.

[2]Note that the ESP specification already discusses facilities for traffic normalization but only supports basic packet padding and considers fully padded channels as too costly.
[3]`http://www.netperf.org`

Benchmarks	No Padding			Fully Padded		On-Demand
	IP	ESP	TFC	1422	800	$\bar{R} = 10^{-1}$s
LAN Throughput (Mbit/s)	570	201	175	58	75	169
TCP Transaction Rate (Hz)	1756	1462	1364	611	740	532
LAN/WAN Overhead (%)	0	10	13	(73)	(61)	≈24
Relative Throughput (%)	283	100	87	28	37	84

Table 3: Throughput and transaction rate for regular and modified IPsec VPN.

We list the overall performance results in Table 3. The first two columns show the testbed performance for raw IP (plain-text) transmission and IPsec ESP tunneling. With 570 Mbit/s, the raw transmission does not reach the expected GBit throughput, likely due to deficient hardware or drivers. As the LAN hosts and the MITM measure the same IP payloads, there is no LAN/WAN overhead. With 201 Mbit/s, the throughput of a standard IPsec ESP tunnel is already notably slower due to 10% protocol overhead but mainly computational constraints of the VPN gateways. As our covert channel mitigation is an extension of this ESP tunnel configuration, we normalize the relative throughput to 100%.

For reference and confirmation of the expected implementation overhead of our prototype, we next evaluated the raw performance of our HPCM Engine compared to the standard IPsec ESP tunnel. The third column "TFC" of Table 3 lists the achieved network performance when tunneling TFC inside ESP with with all covert channel mitigation techniques *disabled*. The overall LAN/WAN overhead of 13% (or 3% when compared with the ESP tunnel) is the result of the 8 to 12 byte TFC protocol encapsulation plus some computational overhead.

4.3 Covert Channel Mitigation Performance

We now describe the behavior and performance of different mitigation policies.

The fourth and fifth column of Table 3 show the performance of a "fully padded channel", enforcing packet sizes of 1422 and 800 bytes at the maximum possible packet rate. For this purpose, we first measured the maximum bi-directional throughput of the VPN channel (201 Mbit/s per direction) and then selected the desired packet rate (inverse IPD) such that the bidirectional channel capacity is almost[4] saturated. We then again measured the maximum (uni-directional) throughput and roundtrip time. As shown in Table 3, the fully padded channel configuration achieves rather poor performance in both configurations, reaching only 37% and 28% of the ESP tunnel throughput. Observe that the enforcement of 800 byte packet size achieves higher transaction rate as well as higher throughput. We believe this is due to the overhead of padding TCP acknowledgements to maximum packet size.

We have also implemented and tested an instantiation of our on-demand mode security management scheme presented in Section 3.2.3. As shown in the last column of Table 3, the employed mode adaption heuristics reach almost the same maximum throughput as the raw TFC encapsulation without time/size padding (169 Mbit/s vs. 175 Mbit/s). The LAN/WAN overhead is slightly higher (24% vs. 13%) and the transaction rate rather low. The high throughput despite relatively high overhead is explained by the mode adaption behavior: As shown in Figure 4, the WAN channel adapts to the maximum possible throughput, but suffers overhead in the rate increase and especially rate decrease phases. As desired by our design in Section 3.2.3, the main impact of reduced token regeneration rates $\bar{R} \leq 15^{-1}s$ in Figure 4 is the increased overhead in the intervals *between* TCP loads, when the rate is not decreased to save tokens.

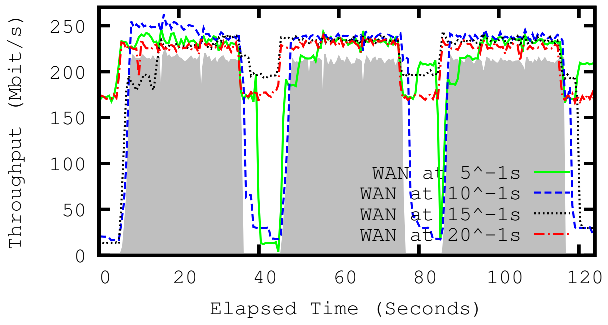

Figure 4: WAN adaption to repeated TCP load.

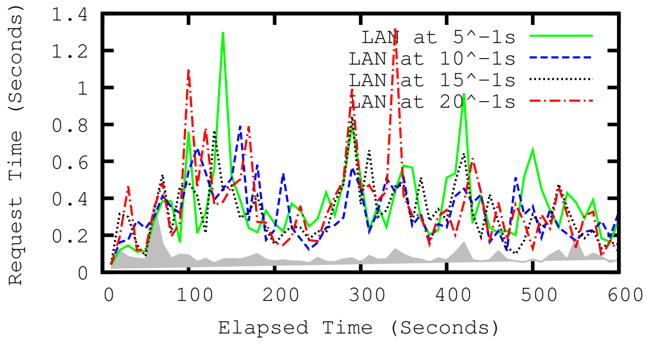

Figure 5: HTTP request delay in mixed traffic.

Finally, we have investigated the ability of our on-demand mode security management to adapt to random, highly heterogeneous traffic patterns one would expect from a VPN with many users. We used Tsung, a traffic load testing tool[5], to record several HTTP sessions in our network, partly also including larger (≈ 60 MB) HTTP downloads. We then configured one of our testbed LANs to act as Internet gateway for the other LAN and used Tsung to replay the recorded HTTP sessions in a pseudo-random fashion with 60 to 80 simultaneous users. Figure 6 shows how the WAN traffic enforcement for four different token regeneration rates $\bar{R}$ dynamically adapts to the LAN usage (grey filled). For $\bar{R} \leq 15^{-1}s$, only the larger peaks in LAN usage influence the WAN traffic enforcement, reducing information leakage

[4]As explained in Section 3.2.1, it is critical that the link is not fully saturated since congestion leads to packet loss and congestion avoidance does not work.

[5]http://tsung.erlang-projects.org

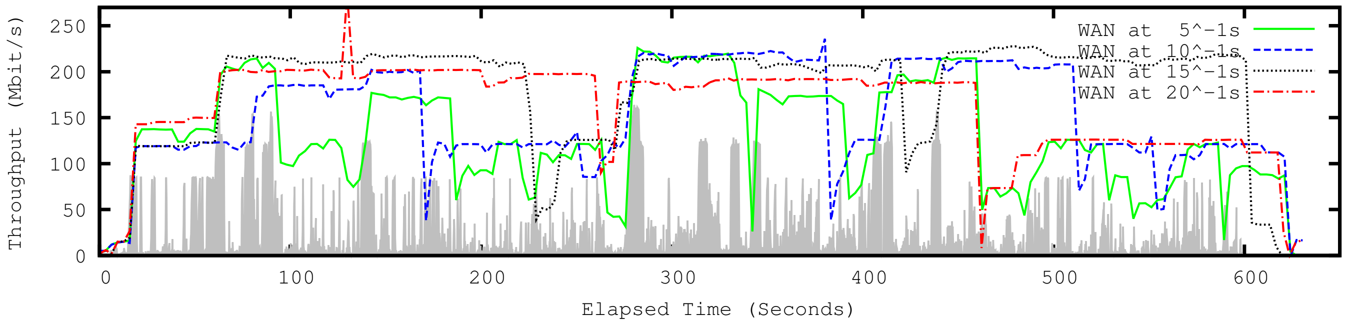

Figure 6: WAN adaption to pseudo-random web traffic and downloads.

at the cost of padding overhead. As shown in Figure 5, the mean duration of responding to individual HTTP requests is kept within reasonable limits. However, in contrast to unpadded traffic (grey filled) the accumulated request delays become noticeable to the user.

In the presented configuration, our mode adaption algorithm switches packet sizes in steps of 100 bytes and packet rates in steps of 1000 packets per second. Considering the maximum WAN packet rate of about 250.000 packets/s, we can derive $C_{r,out}^{\text{ModeSec}} = \bar{R} \cdot \log_2(\frac{1500}{100} \cdot \frac{250000}{1000}) = \bar{R} \cdot 11.87$ and an overall outbound covert channel capacity of, e.g., $\hat{C}_{r,out} = 0.6$ bps for $\bar{R} = 20^{-1}s$ and $N = 1$.

5. RELATED WORK

Several works consider the problem of covert channels and covert channel mitigation in Internet protocols [28,46], yet we know of no works that specially discuss the problem of covert channels in IPsec. The covert channels we identify in IPsec are generally known, but we found no previous discussion of the PMTUD channel. Additionally, the PktSize [18], PktSort [12,34] and DestIP [18] characteristics have different impact in IPsec, and the discussion of storage-based covert channels in the IPsec specification [22] proved to be inaccurate.

Although the IPD-based covert channel is generally well-known [18,20,27,33,44], the problem of inaccuracies in timing enforcement during increased system load remained unsolved [13, 19]. We consider this complication in our design in Section 3.1.2 and present a compensation mechanism that detects and compensates unintended timing inaccuracies. Also, while most works simply assume that packets are of constant size [43] or padded to the maximum desired size [18,46], our adoption of multiplexing and fragmentation enables flexible packet size enforcement. The combination of different mitigation techniques makes our implementation the first prototype for comprehensive covert channel mitigation.

Regarding performance trade-offs, Mode Security was proposed as a general approach to adapt to resource usage by switching between different operation modes [7]. A similar approach called Traffic Stereotyping was proposed for networks [18]. To our knowledge, there is only one system that uses Mode Security to optimize covert channel mitigation, which aims to provide sender anonymity based on dynamic re-routing and IPD enforcement [43,44]. They assume a trusted network stack on each network endpoint and a periodic global negotiation to achieve an equalized traffic matrix [43]. A performance analysis was done based on statistics collected from a medium-sized network [42]; however, no actual performance measurements of their system have been provided and the problem of determining the optimal enforcement mode was left unsolved. Alternatively, NetCamo [20] requires its endpoints to explicitly request their delay and throughput demands beforehand. We extend on these works by proposing a practical algorithm to determine the optimal operation mode on-demand. As we do not aim for sender-anonymity, we do not require mix-networks and various attacks on mixes do not apply to our approach (e.g., [1,41]).

In contrast to probabilistic traffic obfuscation schemes such as HTTPOS [30] or Traffic Morphing [45], our framework enforces an information-theoretic boundary for the maximum information leakage. As argued in Section 1, covert channel detection schemes such as [4,17] are complementary to our work and should be used where mitigation is costly, e.g., for the PktSort and PktDrop characteristics.

While we know of no practical performance measurements for comprehensive covert channel elimination, an overhead of 45%-56% was reported solely for obfuscating the packet size in website traffic [26,45].

6. CONCLUSION AND FUTURE WORK

We have motivated the problem of covert channels in Virtual Private Networks (VPNs) and presented the design, implementation, and performance of a covert channel-resilient VPN. We identified several covert channels and presented new countermeasures. We have investigated the problem of on-demand adaption of operation modes and presented an implementation for comprehensive, high-performance covert channel mitigation in the Linux IPsec stack. Our evaluation shows that on-demand rate adaption is feasible and practical even for highly unpredictive traffic. In more predictable throughput benchmarks, our system achieves remarkable 169 Mbit/s in a 201 Mbit/s VPN connection (84%).

As part of our future work, we will consider the effectiveness of alternative trade-off and normalization strategies. Furthermore, we aim to investigate the impact of inaccuracies in IPD enforcement.

7. ACKNOWLEDGEMENTS

We thank Amir Herzberg, Haya Shulmann and Thomas Schneider for their insightful comments and review of earlier versions of this work.

8. REFERENCES

[1] T. Abraham and M. Wright. Selective cross correlation in passive timing analysis attacks against Low-Latency mixes. In *Global Communications Conference (GLOBECOM)*, pages 1–5. IEEE, Dec. 2010.

[2] K. Ahsan. Covert channel analysis and data hiding in TCP/IP. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, 2002.

[3] D. E. Bell. Looking back on the Bell-LaPadula model. In *Computer Security Applications Conference (ACSAC)*. IEEE, Dec. 2005.

[4] V. Berk, A. Giani, and G. Cybenko. Detection of covert channel encoding in network packet delays. Technical Report TR536, Dartmouth College, Aug. 2005.

[5] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309, Apr. 1998.

[6] B. Briscoe. Tunnelling of Explicit Congestion Notification. RFC 6040, Nov. 2010.

[7] R. Browne. Mode security: An infrastructure for covert channel suppression. In *Research in Security and Privacy (S&P)*, pages 39–55. IEEE, May 1994.

[8] J. Carapinha, P. Feil, P. Weissmann, S. Thorsteinsson, c. Etemoğlu, O. Ingthórsson, S. Çiftçi, and M. Melo. Network Virtualization - Opportunities and Challenges for Operators. In *Future Internet Symposium (FIS'10)*, LNCS, pages 138–147. Springer, 2010.

[9] L. Catuogno, A. Dmitrienko, K. Eriksson, D. Kuhlmann, G. Ramunno, A.-R. Sadeghi, S. Schulz, M. Schunter, M. Winandy, and J. Zhan. Trusted Virtual Domains - design, implementation and lessons learned. In *International Conference on Trusted Systems (INTRUST)*. Springer, Dec. 2009.

[10] Cohesive Flexible Technologies. VPN-Cubed. `http://cohesiveft.com`, Apr. 2012.

[11] J. P. Degabriele and K. G. Paterson. On the (in)security of IPsec in MAC-then-encrypt configurations. In *Computer and Communications Security (CCS)*, pages 493–504. ACM, Oct. 2010.

[12] A. El-Atawy and E. Al-Shaer. Building covert channels over the packet reordering phenomenon. In *International Conference on Computer Communications (INFOCOM)*, pages 2186–2194. IEEE, Apr. 2009.

[13] X. Fu. *On Traffic Analysis Attacks and Countermeasures*. PhD thesis, Texas A&M University, Dec. 2005.

[14] X. Fu, B. Graham, R. Bettati, and W. Zhao. Active traffic analysis attacks and countermeasures. *International Conference on Computer Networks and Mobile Computing*, pages 31–39, 2003.

[15] X. Fu, B. Graham, R. Bettati, and W. Zhao. On effectiveness of link padding for statistical traffic analysis attacks. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 340–349. IEEE, 2003.

[16] J. Gettys. Bufferbloat: Dark buffers in the Internet. *IEEE Internet Computing*, 15(3):96, May 2011.

[17] P. A. Gilbert and P. Bhattacharya. An approach towards anomaly based detection and profiling covert TCP/IP channels. In *Information, Communications and Signal Processing (ICICS)*, pages 1–5. IEEE, Dec. 2009.

[18] C. G. Girling. Covert channels in LAN's. *IEEE Transactions on Software Engineering*, 13(2):292–296, Feb. 1987.

[19] B. Graham, Y. Zhu, X. Fu, and R. Bettati. Using covert channels to evaluate the effectiveness of flow confidentiality measures. In *Parallel and Distributed Systems (ICPADS)*, pages 57–63. IEEE, 2005.

[20] Y. Guan, X. Fu, D. Xuan, P. U. Shenoy, R. Bettati, and W. Zhao. NetCamo: Camouflaging network traffic for QoS-guaranteed mission critical applications. *Trans. on Systems, Man, and Cybernetics - Systems and Humans*, 31(4):253–265, July 2001.

[21] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303, Dec. 2005.

[22] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301, Dec. 2005.

[23] C. Kiraly, S. Teofili, R. Lo Cigno, M. Nardelli, and E. Delzeri. Traffic flow confidentiality in IPsec: Protocol and implementation. In *The Future of Identity in the Information Society*, pages 311–324. Springer, June 2008.

[24] D. Kundur and K. Ahsan. Practical internet steganography: Data hiding in IP. In *Texas Workshop on Security of Information Systems*, Apr. 2003.

[25] B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[26] M. Liberatore and B. N. Levine. Inferring the source of encrypted HTTP connections. In *Computer and Communications Security (CCS)*, pages 255–263. ACM, Nov. 2006.

[27] Y. Liu, D. Ghosal, F. Armknecht, A.-R. Sadeghi, S. Schulz, and S. Katzenbeisser. Hide and seek in time — robust covert timing channels. In *European Symposium on Research in Computer Security (ESORICS)*, volume 5789 of *LNCS*, pages 120–135. Springer, Sept. 2009.

[28] D. Llamas, C. Allison, and A. Miller. Covert channels in internet protocols: A survey, Mar. 2006.

[29] N. Lucena, G. Lewandowski, and S. Chapin. Covert channels in IPv6. In *Privacy Enhancing Technologies*, chapter 10, pages 147–166. Springer, 2006.

[30] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Network and Distributed Systems Security (NDSS)*. Internet Society, 2011.

[31] J. Millen. 20 years of covert channel modeling and analysis. In *Research in Security and Privacy (S&P)*, pages 113–114. IEEE, May 1999.

[32] J. Mogul and S. Deering. Path MTU discovery. RFC 1191, Nov. 1990.

[33] I. S. Moskowitz and A. R. Miller. Simple timing channels. In *Research in Security and Privacy (S&P)*,

pages 56–64. IEEE, May 1994.

[34] S. Murdoch and S. Lewis. Embedding covert channels into TCP/IP. In *Information Hiding*, chapter 19, pages 247–261. Springer, 2005.

[35] National Computer Security Center. *A Guide to Understanding Covert Channel Analysis of Trusted System*, Nov. 1993.

[36] C. Perkins. IP Encapsulation within IP. RFC 2003, Oct. 1996. Updated by RFC 3168.

[37] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, Sept. 2001. Updated by RFCs 4301, 6040.

[38] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Computer and Communications Security (CCS)*, pages 199–212. ACM, Nov. 2009.

[39] A.-R. Sadeghi, S. Schulz, and V. Varadharajan. The silence of the LANs: Efficient leakage resilience for IPsec VPNs (full version). Technical report, June 2012.

[40] S. D. Servetto and M. Vetterli. Communication using phantoms: Covert channels in the internet. In *International Symposium on Information Theory*, page 229. IEEE, 2001.

[41] V. Shmatikov and M.-H. Wang. Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses. In *European Symposium on Research in Computer Security (ESORICS)*, LNCS, pages 18–33. Springer, Sept. 2006.

[42] B. R. Venkatraman and R. E. Newman-Wolfe. Performance analysis of a method for high level prevention of traffic analysis using measurements from a campus network. In *Computer Security Applications Conference (ACSAC)*, pages 288–297. IEEE, Dec. 1994.

[43] B. R. Venkatraman and R. E. Newman-Wolfe. Transmission schedules to prevent traffic analysis. In *Computer Security Applications Conference (ACSAC)*, pages 108–115. IEEE, Dec. 1994.

[44] B. R. Venkatraman and R. E. Newman-Wolfe. Capacity estimation and auditability of network covert channels. In *Research in Security and Privacy (S&P)*, pages 186–198. IEEE, May 1995.

[45] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Network and Distributed Systems Security (NDSS)*. Internet Society, 2009.

[46] S. Zander, G. Armitage, and P. Branch. A survey of covert channels and countermeasures in computer network protocols. *Comm. Surveys & Tutorials*, 9(3):44–57, 2007.

[47] W. Zhao, D. Olshefski, and H. Schulzrinne. Internet quality of service: An overview. Technical report, Columbia University, 2000.

APPENDIX

A. COVERT CHANNELS IN IPSEC

In general, for a channel characteristic to be exploited as a covert channel, it must be measurable after transformation by the VPN gateway. An outbound covert channel requires that certain characteristics of packets remain measurable when they are encapsulated and encrypted when traversing from the LAN to the WAN side. Similarly, inbound covert channels require packet characteristics to survive the decapsulation from the WAN to the LAN side.

In this section we review the impact of covert channels on VPNs that apply to the setup described in Section 2. Many of the channels are known, however, their impact is different in VPNs and other mitigations are possible.

Unfortunately, no systematic approach exists for identifying covert channels besides systematic exhaustive search [35]. The following results are based on such examination of the IPsec specification [22], which standardizes the process of encapsulation and decapsulation at the VPN boundary, and related work. We then examined recent Linux and OpenBSD implementations to validate[6], exposing notable deviations from the specification.

A.1 Storage Channels

Several storage-based covert channels have been identified in TCP/IP networks [28,29,46]. Storage-based covert channels are usually easy to eliminate, but create reliable and efficient channels when left undetected.

A.1.1 Differentiated Services (DS)

The 6 bit DS field in the IP header is used to signal service requirements, such as realtime VoIP traffic. Hence it may leak information about the type of the transmitted data even if not actively exploited by an insider. The IPsec specification recognizes this problem, however, countermeasures are only described for *inbound* covert channels [22]. The proposed countermeasure is to discard the DS field of the outer IP header during defragmentation, unless explicitly configured to do otherwise. Indeed, the examined Linux implementation honors this recommendation, while OpenBSD always discards the outer header's DS field during decapsulation. Hence we get $C^{\mathrm{DS}}_{m,out} = 6$ bpp in both cases (and likely also for most other IPsec implementations) while the optional decapsulation of the DS field opens another inbound channel of the same capacity $C^{\mathrm{DS}}_{m,in} = 6$ bpp.

A.1.2 Explicit Congestion Notification (ECN)

ECN uses two bits of the IP header to let routers signal network congestion to the endpoints. As such, ECN between WAN routers and LAN endpoints may directly violate the VPN policy. Originally, a "limited functionality" mode was supplied to eliminate such covert channels in IPsec, which effectively precludes the WAN from ECN signaling [37]. However, the current revision of IPsec specifies that the ECN field should be copied during encapsulation, and one bit is copied back to the inner header on decapsulation, if ECN is enabled on the inner IP header [22]. As such, the specification enables covert channels with a capacity of $C^{\mathrm{ECN}}_{m,in} = 1$ and $C^{\mathrm{ECN}}_{m,out} = 2$ bpp. The ECN treatment was refined and unified in [6]. After discussing the trade-off between covert channels and ECN, an updated scheme is defined with $C^{\mathrm{ECN}}_{m,out} = 2$ bpp and $C^{\mathrm{ECN}}_{m,in} = 1.5$ bpp. Furthermore, a legacy-compatible mode with $C^{\mathrm{ECN}}_{m,out} = 0$ but $C^{\mathrm{ECN}}_{m,in} = 1.5$ is specified. The inbound covert channel is deemed insignificant [6].

[6]We examined IPsec implementations of recent Linux 2.6.36 and OpenBSD 4.8.

The examined Linux and OpenBSD systems both implement the original ECN specification [37]. They can be configured to use "limited functionality" mode, eliminating ECN-based covert channels. However, Linux does not drop packets where the outer ECN field was invalidly set to *Congestion Experienced* (see also Section A.2.4).

A.1.3 IPv4 Flags (Flags)

The IPv4 Flags field consists of one unused bit (*Reserved*) and two bits for fragmentation, Don't Fragment (DF) and More Fragments (MF). The IPsec specification discusses the problem of fragmentation at great length, but refers to standard IP-IP encapsulation [36] for the exact treatment of inner and outer header fields. In general, the IPsec gateway must copy the DF bit if set by the sender to enable Path MTU Discovery (PMTUD) along the route. Otherwise, if the sender allows fragmentation, the IPsec gateway is is advised to still use PMTUD but fragment the original packet before encapsulation. If the advise is followed, no outbound covert channel is created by the Flags field. During decapsulation, any WAN fragments must first be reassembled, eliminating any option for the adversary to encode information for the final recipient of the packet.

The examined OpenBSD implementation correctly resets the DF and MF bits on encapsulation and decapsulation, but copies the *Reserved* bit as a side-effect of this normalization. The Linux implementation copies the DF bit by default unless PMTUD has been disabled by configuration. However, in this case a compatibility issue leads to a repeated incrementation of the IP ID field for fragmentable packets. Hence, both implementations leak information about the Flags field settings on the inner header, creating an outbound covert channel with $C_{m,out}^{\text{Flags}} = 1$ bpp.

A.1.4 Destination IP (DestIP)

The destination IP address can be used as a covert channel [18]. Although the insider in a VPN cannot directly modify the destination address visible in the WAN, the destination IP can be modified indirectly if more than to LAN sites are connected in the VPN, by addressing the packet to one or the other LAN site. However, and assuming that other covert channels are eliminated, the MITM cannot distinguish the (time and size padded) stream of packets for each VPN channel to determine which channel is being preferred by the insider. Hence, the characteristic only acts as an *amplifier* for an existing outbound covert channel, multiplying the symbol space of the existing covert channel by the number of alternate destination LANs N.

A.2 Timing Channels

A.2.1 Packet Size (PktSize)

Information can also be encoded in the size of packets [18]. In IPsec VPNs, the created covert channel is a limited outbound channel since (1) the MITM cannot covertly modify the packet size and (2) the symbol space is reduced by ESP payload alignment and block cipher padding [21]. To estimate the available symbol space, consider that the maximum and minimum possible packet length is reduced by protocol headers. The remaining possible packet lengths are then partitioned due to ESP padding and alignment. For example, the maximum packet size in standard Ethernet LAN is limited by the 1500 byte Maximum Transmission Unit (MTU), minus 40 bytes inner and outer IP headers and two times ≈ 12 bytes for the ESP header with cipher block padding and ESP MAC, respectively. Hence $l_{max} \approx 1500 - 2 \cdot 20 - 2 \cdot 12 = 1436$ bytes and $l_{min} \approx 2 \cdot 20 + 2 \cdot 12 = 64$[7]. Considering the 4 byte alignment of ESP we get $C_{m,out}^{\text{PktSize}} = \log_2(\frac{1436-64}{4}) = 8.4$ bpp.

A.2.2 Inter-Packet Delay (IPD)

Several previous works discuss exploitation and mitigation techniques for covert channels based on the relative delay between packets (IPD) [4, 18, 20, 27, 33]. The characteristic can be used for both, outbound and inbound covert channels and is not significantly affected by IPsec processing. The channel capacity generally depends on the accuracy of the timing measurements on the receiver [33]. However, when approximating it as simple binary channel that either delays a packet or not, it achieves a capacity of $C_m^{\text{IPD}} = 1$ bpp. Previously, a throughput of 0.98 bpp was reported on an intercontinental Internet connection [27].

A.2.3 Packet Order (PktOrd)

Information can also be encoded by changing the order of packets [12, 24]. In IPsec VPNs, the MITM cannot distinguish the order of IPsec payloads. However, the MITM may reorder WAN packets to send information to an insider, creating an inbound covert channel. The receiving IPsec gateway can optionally maintain a partial anti-replay window, a bit mask that marks the last r received packets based on the Encapsulated Security Payload (ESP) or AH sequence number field. If used, a typical replay window of, e.g., $r = 32$ packets leaves an inbound covert channel capacity of $C_{m,in}^{\text{PktOrd}} = 1/r \cdot \log_2(r!) = 3.67$ bpp.

A.2.4 Packet Drops (PktDrop)

Packet dropping was proposed in [40] to create covert channels. However, since the MITM in a VPN cannot distinguish secure channel packets, the outbound version of this channel is equivalent to the IPD-based covert channel. When used as an inbound covert channel, one LAN client may generate a stream of enumerated packets from which the WAN MITM adversary may drop some at will. Hence, a reliable covert channel with at most $C_{m,in}^{\text{PktDrop}} = 1$ bpp (drop/no drop) can be created.

A.2.5 Path MTU Discovery (PMTUD)

PMTUD is a mechanism to dynamically detect the Path MTU (PMTU), the maximum allowed packet size along an IP path [32]. By setting the Don't Fragment (DF) flag in the IP header, the sender asks intermediate routers to not fragment packets on demand but to drop it and instead report the MTU of their network segment as an ICMP error (*PMTU error*). The sender notes the respective smallest reported MTU towards a particular destination as the PMTU and adjusts the size of emitted packets accordingly. The sender will also periodically test by trial&error if a larger PMTU is possible, e.g., if the routing was changed (PMTU aging).

While the size of VPN packets cannot be manipulated directly, the MITM may artificially limit the MTU of his network segment to inject information via PMTUD. Specifically, when using PMTUD in the WAN, the MITM may

[7]Excluding the block cipher's initialization vector (IV).

inject ICMP errors to reduce the PMTU perceived by the IPsec gateways. Upon receiving a packet that is larger than the known PMTU for the respective WAN destination, and if that packet has the DF bit set, IPsec gateways will propagate the smaller PMTU to the respective LAN client by synthesizing the appropriate PMTU error [22].

Since the PMTU is unlikely to change frequently, the IPsec specification recommends that gateways follow the PMTU aging process described in the PMTUD specification [32]. There, a periodic timeout of $t = 2$ minutes is recommended before attempting to increase the PMTU, and immediately increasing further if the attempt was successful, until the new PMTU is discovered. Since only smaller PMTU values are propagated to LAN clients, this limits the rate of PMTU errors that the MTIM can usefully inject.

For example, if we assume possible MTU range from the minimum IP packet size (768 bytes) to the maximum Ethernet MTU minus IPsec overhead ($\approx 1500 - 20 - 35 = 1445$ bytes), with changes in 4 byte steps due to ESP padding and alignment, there are $M \approx \frac{1445-768}{4} \approx 169$ different packet sizes that could be reported by the IPsec gateway to the LAN client. Furthermore, the adversary can mitigate the delay imposed by the PMTU aging timeout t by partitioning the symbol space such that at most $M/2 = 169/2 = 84$ subsequent 1 bit symbols can be sent, achieving up to $C^{\text{PMTUD}}_{m,in} = \frac{M/2}{t} = 842 \cdot 60 = 5.18$ bps.

A.3 Active Probing

The MITM may infer information on the LAN status by actively probing the IPsec gateways and evaluating their response behavior, an approach that was introduced as active traffic analysis [13, 14]. In particular, a LAN client could cause high load on the LAN interface of the IPsec gateway. The resulting change in the gateway's system load can then be measured by how it responds to legitimate service requests by the MITM, such as ICMP pings [13].

Note that, contrary to the previous channel characteristics, this attack actually exploits a side channel at the gateway: Its capacity does not depend on the usage of the VPN channel but on the frequency at which the insider can induce high and low system loads at the gateway as well as on the rate at which the MITM is able to probe the gateway to measure its system load with sufficient accuracy.

It was previously proposed to either normalize the (seemingly uncritical) responses by the gateway. We also believe that the attack can be prevented by combining the IPsec gateway with a second physical firewall on the WAN side, to filter invalid requests and act as key negotiation server on behalf of the VPN gateway. However, such side-channels are outside the scope of this work.