Systematic Engineering of Control Protocols for Covert Channels

Steffen Wendzel and Jörg Keller

University of Hagen
Faculty of Mathematics and Computer Science
58084 Hagen, Germany
swendzel@ploetner-it.de, joerg.keller@FernUni-Hagen.de

Abstract. Within the last years, new techniques for network covert channels arose, such as covert channel overlay networking, protocol switching covert channels, and adaptive covert channels. These techniques have in common that they rely on covert channel-internal control protocols (so called micro protocols) placed within the hidden bits of a covert channel's payload. An adaptable approach for the engineering of such micro protocols is not available. This paper introduces a protocol engineering technique for micro protocols. We present a two-layer system comprising six steps to create a micro protocol design. The approach tries to combine different goals: (1) simplicity, (2) ensuring a standard-conform behaviour of the underlying protocol if the micro protocol is used within a binary protocol header, as well as we provide an optimization technique to (3) raise as little attention as possible. We apply a context-free and regular grammar to analyze the micro protocol's behavior within the context of the underlying network protocol.

Keywords: Network Covert Channel, Covert Channel Control Protocol

1 Introduction

A covert channel is a communication channel that was not designed to be used for a communication [9]. Such covert channels can occur in computer networks and can be used by attackers to overcome security policies (e.g. transferring illicit information or botnet traffic [25]). Covert channels are basically divided into two classes, covert *timing* channels and covert *storage* channels [14]: Timing channels are based on the manipulation of timing and sorting behavior (e.g. timing of network packets), and storage channels use storage attributes (e.g. unused bits within a network packet's header) to transfer hidden information.

Since the 1980's, a number of different network covert storage channels in protocol headers were discovered and evaluated, such as in LANs [5], IP and TCP [4, 15, 18], IPv6 [12], ICMP [17] and HTTP [2]. These covert channel techniques are usually based on the idea to utilize unused/reserved header areas or on the idea to modify currently not required header areas (such as fragmentation bits or the least significant bits of the TTL in IPv4).

Later, covert channel-internal control protocols, so called *micro protocols*, were introduced [17]. These protocols are used to implement features such as reliability as well as connection management into covert channel software. Using micro protocols, covert channel systems can build hidden overlay networks [23], and can adapt to changes in the underlay network (e.g. changing firewall rules) [10, 24]. Up to now, micro protocols were designed in an ad hoc manner.

We present a systematic approach to design micro protocols suitable for binary network protocols (i.e. no plaintext protocols), including protocols which were already in the scope of previous covert channel research (such as IP), but also protocols which have not been part of a covert channel investigation so far (such as link control protocol (LCP) bits in PPP and future protocols). The result of the approach is an implementation-ready protocol design, i.e. our method focuses on the protocol design and protocol implementation phase of protocol engineering. The method is designed to be simple and thus comprises only six steps. It ensures that if the resulting micro protocol is used, the behaviour of the underlying network protocol is still standard-conform. We also minimize the attention raised by the micro protocol by optimizing the mapping of micro protocol bits to the bits of the underlying protocol.

The remainder of this paper is structured as follows. Section 2 describes the protocol engineering method in detail and also provides an exemplary walkthrough for a simple example protocol. Section 3 discusses the results of our work and a conclusion is provided in Section 4.

2 Micro Protocol Engineering

This section presents a design method that optimizes the hidden placement of a covert channel-internal protocol (the so-called *micro protocol*) within the utilized area of an underlying network protocol. Therefore some bits of the underlying protocol are selected to carry covert data. The bit values of the micro protocol are mapped to bit values of the selected area in a way that reflects the rules of the underlying protocol (e.g. standard conform behavior). The mapping also aims to minimize the attention raised by the covert channel.

Selection techniques for network covert storage channels were already presented in the past. For instance, Li et al. developed an approach that uses the concept of natural selection to determine suitable network protocols which can be used for a covert communication [10] and Yarochkin et al. presented similar work to evaluate utilizable network protocols based on passive network monitoring [24].

Covert channel optimization with the goal to minimize the raised attention of the data transfer was already presented by Wendzel and Keller in [23]. The paper discussed the usage of multiple network protocols for a covert channel within a mobile context. The cooperative optimization between covert channel participants was realized using a micro protocol. We extend the work of [23] in this paper with the optimization and validation aspect of micro protocols.

Before we will describe our six step approach in detail, we explain our terminology and goals.

2.1 Terminology

We use three readily understandable terms to describe our approach:

- The **underlying protocol** is the protocol that is utilized by the covert channel. For instance, if the covert channel utilizes some bits of the IPv4 header, the underlying protocol is IPv4.
- We combine the utilized areas of the underlying protocol as proposed in [23]. We call the combined areas the **cover protocol**. For instance, if the two least significant bits of the IPv4 TTL as well as the reserved flag are utilized, these three bits are called the cover protocol. We use this term to help the reader to distinguish between the underlying protocol itself and the area used for the covert operation.
- The **micro protocol** is placed within the cover protocol (i.e. within the hidden data) and internally controls the covert channel.

2.2 Goals

Besides the major goal, to enable a protocol designer to create a suitable micro protocol, several other important goals exist. The result of the approach will be a verified protocol design ready for an implementation.

First of all, one goal was to *achieve a simple approach*. Therefore, we decided to develop a two-layer system. One layer represents the utilization of bits within the underlying network protocol, the other layer represents the micro protocol mapping.

Part of the layered approach is to ensure the *standard-conform behavior of the underlying network protocol*. Covert channels can break standard-conform behavior by setting bit combinations which are not valid, if they do not take the rules of the underlying protocol into account. For instance, a covert channel could set both, the *SYN flag* and the *RST flag* of TCP, within the same packet. Other typical incorrect bit combinations for TCP/IP protocols can be found in the documentation of normalizer systems like *norm* [7], OpenBSD's *pf scrubbing* [16], or the *Snort* normalizer [20]. Not only can normalizers drop or modify headers which are not standard-conform, those headers can also raise attention and make a monitor aware of the covert channel's existence.

The third goal is to *minimize the attention raised by the micro protocol*, i.e. to stay hidden, what is again done by optimizing the mapping of micro protocol bits to bits of the cover protocol. Taking the mentioned standard-conformity (2nd goal) into account ensures a low profile as well.

Another goal of our approach is to provide a *dynamic re-designing* and *optimization framework* by enabling the protocol designer to repeat only selected steps of our six step approach. For instance, it is possible that the designing process at one of the six steps motivates the designer to choose the three least

significant bits of the TTL instead of the two least significant bits of the TTL with the reserved flag. Therefore we must enable the designer to go back to an earlier step of the approach in a way that the work already done at later stages of the design process is still useful.

Last but not least, the approach shall be usable for all binary network protocol headers (e.g. IPv4, IPv6, ARP, TCP). As we will discuss in Sect. 3, the approach is not intended for plaintext protocols (such as HTTP).

2.3 Six-Step Approach

We present a 6-step method (visualized in Fig. 1) to create a micro protocol design. The micro protocol's behavior is linked to the underlying protocol's behavior. We achieve the previously mentioned goals by first evaluating the probability of the possible values of the underlying protocol's bits, and secondly, by evaluating the standard-conform behavior of the underlying protocol in case the micro protocol is used. If the behavior is not standard-conform, we propose several modifications which will result in a better configuration. The process is designed to be incremental. The design flow stops after the 6th step but can be repeated in various steps if the result is not satisfying, as we discuss in Sect. 2.6. The result of the engineering approach is an implementation-ready protocol design. It is important to understand that in protocol engineering, the *testing* phase verifies the functionality of a software and thus depends on the *implementation* phase [8], which is not in our scope. Therefore, we focus on the *validation* phase (step 6) to verify the logical correctness of the design.

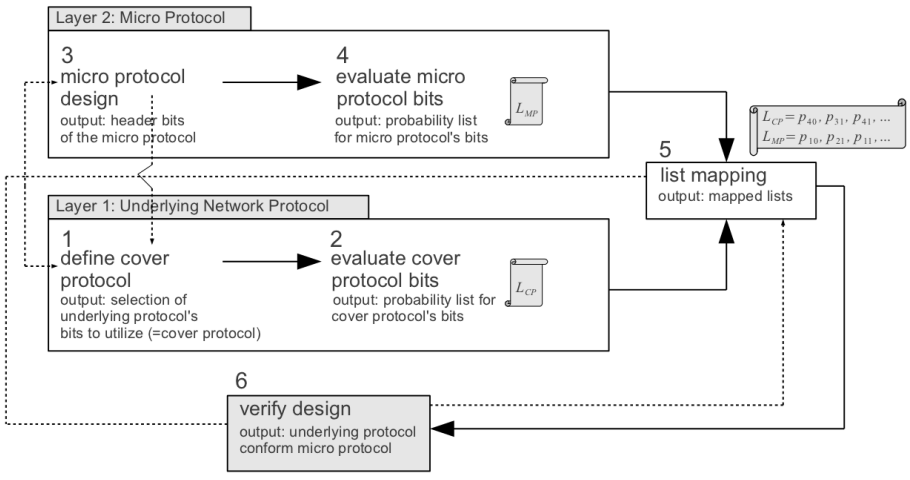

Fig. 1. The two-layer micro protocol engineering approach. Dashed arrows represent possible re-engineering paths.

Step 1) Define the cover protocol: Before a micro protocol can be placed in an underlying network protocol, areas of the underlying network protocol must be selected which can be used to place hidden information into. As discussed earlier, we call the selected area of the underlying network protocol the *cover protocol*.

The utilizable areas for the cover protocol can be figured out in two ways:
1) If an underlying protocol is to be used that was already part of a covert channel investigation (this applies for most important protocols, such as IP, IPv6, ICMP, HTTP, and LAN protocols, as mentioned in Sect. 1), the existing literature can be taken into account (e.g. Rowland utilizes the Identifier of IPv4 and the ISN of TCP [18], while Lucena et al. evaluate parts of the IPv6 header for covert information transfer [12]). This first case has the advantage that for many protocols previous work is available concerning the detectability and prevention of different covert channel variants. For network covert storage channels, traffic normalization turned out to provide the best protection, since more than 80 storage channels can be eliminated by today's normalizers [22]. However, as detection and protection of covert channels are complex topics, we cannot go into details.
2) If, on the other hand, a protocol is to be used for a covert operation that was not part of a previous investigation, the analysis of utilizable (e.g. non-required or reserved) areas of the header needs to start from scratch. This step requires access to a detailed protocol documentation, and optionally to a protocol implementation. In most situations, the RFC document that describes the protocol will be the most important source for this step.

Step 2) Evaluate selected cover protocol areas: The probability of each bit i in the cover protocol having value j, i.e. $p_{ij} = prob(b_i = j)$ is evaluated as proposed in [23]. Note that the probabilities for different bits are not necessarily independent (e.g. the valid ICMP codes depend on the ICMP type value). Thus, if the number n of bits is small, we might alternatively take the probabilities for all possible bit patterns of the cover protocol. This increases the number of probability values from $2n$ to 2^n. If this is not possible, the dependencies are taken care of in the last step. To gain such information, relative frequencies from traffic recordings can be evaluated. If it is not possible to define exact numeric values for the probability values of the cover protocol's bits, a small classification set (e.g. $\{low, medium, high\}$) as also mentioned in [23] can be used. Afterwards, all bits with all possible values are sorted in a list L_{CP} in increasing order of probability. For instance, if a cover protocol contains only two bits b_0 and b_1 with the values $p_{00} = 0.01, p_{01} = 0.99, p_{10} = 0.5, p_{11} = 0.5$, then L_{CP} will be $(p_{00} \rightarrow p_{10} \rightarrow p_{11} \rightarrow p_{01})$ or $(p_{00} \rightarrow p_{11} \rightarrow p_{10} \rightarrow p_{01})$.

Step 3) Micro protocol design: The micro protocol M that is to be implemented within the cover protocol C is required to be designed within the 3rd step. It is necessary that $sizeof(M) \leq sizeof(C)$ since only $sizeof(C)$ bits are available. If it turns out that more bits are required than available, either the micro protocol has to be re-designed, or steps 1 and 2 are required to be repeated until enough space is available within the cover protocol C.

The micro protocol design contains the task to build the micro protocol's header which involves to define the required bits (e.g. an ACK flag, sequence numbers, or flags to start/end a transaction). A first micro protocol was defined by Ray and Mishra in 2008 [17], but contains a fixed size, i.e. it is not adaptable. A space-inefficient protocol can also be found in the tool *pingtunnel* [21]. However, it is up to the designer to choose required functionality (e.g. a "disconnect" flag) for the micro protocol that will build the required micro protocol header fields. Typical means such as UML modeling, Petri nets, SDL and LOTOS can be applied in this step if the protocol becomes complex. However, if the micro protocol is small (as it usually is, since only a very limited space is provided within a cover protocol), these means are not mandatory.

Step 4) Evaluate the micro protocol: This step is similar to step 2 but evaluates the probability values for the bits of the micro protocol. For new protocols, traffic recordings are not available. In this case, relative frequencies of protocol messages must be guessed to derive bit frequencies or bit pattern frequencies. The result is a second list L_{MP} sorting the frequencies of the bit values in the micro protocol.

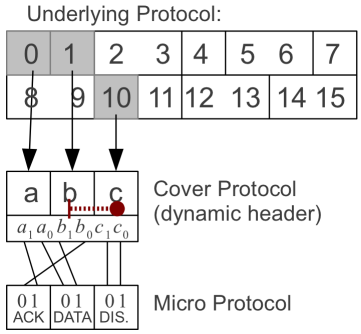

Fig. 2. A sample underlying protocol with three utilized bits building the cover protocol. A sample mapping of the micro protocol bits to the cover protocol bits is additionally shown. We assume, "c" is only valid if "b" is set.

Step 5) List mapping: Finally, both lists are mapped (cf. Fig. 2). If bit patterns are used, the mapping of lists sorted according to probabilities is rank-preserving and thus optimal in the sense that it minimizes the sum of the squares of probability differences before and after the mapping. So far, we have assumed that both lists are equal in length. However, this need not be the case. For instance, if we have a cover protocol providing two bits and a micro protocol requiring only one bit (i.e. two states), we might map the end of both lists, of which the last two values (for the two states) are used:
$L_{CP} = p_{00} \rightarrow p_{11} \rightarrow p_{10} \rightarrow p_{01}$ and $L_{MP} = \emptyset \rightarrow \emptyset \rightarrow p_{01} \rightarrow p_{00}$.
Thus, we map p_{00} of the micro protocol to p_{01} of the cover protocol, as well as p_{01} of the micro protocol to p_{10} of the cover protocol. p_{11} and p_{00} of the cover

protocol are not required to be used for the operation. If one can choose, one will normally map micro protocol values to cover protocol values which have similar probabilities in order to minimize the attention raised by the covert channel.

Step 6) Verify the micro protocol design: In the final step, we apply a verification to ensure all possible micro protocol states can be reached without causing an incorrect behavior of the underlying protocol, e.g. behavior that is not conform to a RFC specification. For instance, if different bits of the ICMP type and code values are utilized by the cover protocol, conditions apply where code values which are valid for ICMP type X are not valid for type Y. Similar situations can apply for the IPv4 "options", the IPv6 "next header", the link control protocol's (LCP) configuration options of the PPP protocol, and other protocols with a dynamic header design. To implement step 6, a context-free or regular grammar (Chomsky type 2 or 3, respectively) can be applied. Formal grammars have been applied to other areas of information security as well, such as for attack modeling [6], and were also used in the area of protocol engineering (e.g. for modelling events and states for protocol implementations [11]).

We propose a two-layer approach for this task (related to the two layers of our whole protocol engineering method). First, the rules for the cover protocol must be defined in the context of the underlying protocol. A subset of the header's bits are used to form the cover protocol (for convenience we do not use the bit numbers but the letters a, b and c of these bits as labels).

To simulate a dynamic header design and to illustrate a reasonable situation, we assume that the bit "c" is only valid if bit "b" is set (cf. Fig. 2). Therefore, we can build the formal grammar $G = (V, \sum, P, S)$, where V is the set of non-terminals, $\sum$ the set of terminals, P the set of productions, and $S \in \sum$ is the start symbol [6]. Chomsky type-2 grammars, the so-called *context-free grammars*, do only allow production rules with a single non-terminal on the left side of a production rule. The right side of a production rule can contain both, terminals and non-terminals. Chomsky type-3 grammars, the so-called *regular grammars*, in comparison to type-2 grammars, additionally restrict the right hand side of a production rule to be either the empty string, a single terminal, or a single terminal symbol and a single non-terminal that is either always on the right or always on the left side of the terminal symbol in each production. We define the cover protocol either as a type-2 or type-3 grammar. A type-3 grammar provides the advantage to ease the final task due to the faster verification of language subsets. A type-2 grammar, on the other hand, makes it easier to find suitable production rules. For the final validation process, it is important, that all grammars generate only sentences where the terminals appear in the order of the actual bits of a header. For instance, a terminal representing the "Don't Fragment" flag in the IPv4 header should not be placed before a terminal representing the "Reserved" flag as long as the "Reserved" flag is part of the cover protocol.

The type-2, i.e. *context-free*, grammar $G_{CP} = (V, \sum, P, S)$, $V = \{S, A, B, C\}$, $\sum = \{a_0, a_1, b_0, b_1, c_0, c_1\}$ for the given example is as follows:

$$P = \{S \rightarrow AB|AC \tag{1}$$
$$A \rightarrow a_1|a_0 \tag{2}$$
$$B \rightarrow b_1|b_0 \tag{3}$$
$$C \rightarrow b_1c_1|Bc_0\} \tag{4}$$

The structure of the regular grammar $G_{CP} = (V, \sum, P, S), V = \{S, B, C_A, C_B\}$, $\sum = \{a_0, a_1, b_0, b_1, c_0, c_1\}$ for the same example is as follows:

$$P = \{S \rightarrow a_0B|a_1B \tag{5}$$
$$B \rightarrow b_0C_A|b_1C_B \tag{6}$$
$$C_A \rightarrow c_0 \tag{7}$$
$$C_B \rightarrow c_0|c_1\} \tag{8}$$

Creating such a grammar for a cover protocol normally is not too difficult. First, it is possible that a grammar-based specification of the underlying protocol is already available or can be obtained easily. Some major RFCs for instance contain state machine models for network protocols which can be translated into a grammar, or the underlying protocol might have been used previously as a cover protocol. Second, a cover protocol typically is small compared to the whole underlying protocol and thus not all header fields must be considered.

In the second step, we write down the mapping of step 5 for the covert channel's micro protocol (micro protocol layer). For instance, $ACK \equiv a_1, \neg ACK \equiv \neg a_0, DATA \equiv b_1, \neg DATA \equiv \neg b_0, DISCON \equiv c_1, \neg DISCON \equiv \neg c_0$.

We build a second grammar G_{MP} for the micro protocol based on these mappings. For this second grammar G_{MP}, the order of terminals must reflect the order of bits in the underlying protocol's header as well to ensure a correct language verification. Thus, even if the bits are in a different order within the micro protocol's header, the sentences of G_{MP} must be built respecting the order of the underlying protocol's header.

Afterwards, we verify whether the language produced by $L(G_{MP})$ is a subset of $L(G_{CP})$. If the language is a subset, the micro protocol matches the conditions of the underlying protocol.

Manual verification:
Since both, cover protocols and micro protocols, are usually small, i.e. only a few bits are available, it is possible to verify whether $L(G_{MP}) \subseteq L(G_{CP})$ by hand. Therefore, it is required to build sentences for all possible conditions of the micro protocol (e.g. setting flag X and flag Y within the same packet). For instance, to test whether the "ACK" flag and the "DIS" flag can be set within the same micro protocol header without breaking the standard conform behavior of the underlying protocol, we have to verify, if the following sentence of G_{MP} within G_{CP} is possible:

$$\{ACK, \neg DATA, DIS\} \equiv a_1b_0c_1 \tag{9}$$

However, the production rules do not allow to create the sentence "$a_1b_0c_1$" (only similar results are possible: "$a_1b_0c_0$" (AC), "$a_1b_1c_1$" (AC) and "$a_0b_1c_1$" (AC)). Thus acknowledging data and introducing a disconnect at the same time within the covert channel connection is not feasible with the provided configuration due to the conflict of setting the bits "a" and "c" without setting the bit "b" (DATA flag). We discuss solutions for this problem in Sect. 2.6.

Automatic verification:
As shown and proven by Baldoni et al. and Bouajjani et al., an automatic verification is feasible, as long as G_{CP} is a regular language (G_{MP} can be a context-free language nevertheless) [1, 3]. The authors present an approach similar to ours, but in a research field other than security where Agent UML (AUML) and DyLOG (a logic programming language based on modal logic) were both transferred into formal grammar to verify whether an implementation is conform to the abstract specification [1].

In contrast to Baldoni et al., we do not focus on the validation of an implementation, but on the validation of a micro protocol inside a cover protocol.

The inclusion of the context-free language A in the regular language B, i.e. $A \subseteq B$, can be verified by computing whether $A \cap \overline{B} = \emptyset$ [1, 3].[1] The result of the intersection of the context-free language A and the regular language $\overline{B}$ is another context-free language [1] for which the emptiness is decidable [19].[2]

An algorithm to verify the containment of a context-free language in a regular language can be found in [3] and is applied in the previously mentioned work of Baldoni et al. [1]. Given G_{MP} and the deterministic automaton for $L(G_{CP})$ as input, $L(G_{MP}) \subseteq L(G_{CP})$ can be calculated in $O(p \cdot s^3)$ time, where p is the number of productions of G_{MP} and s is the number of states of the automaton of $L(G_{CP})$.

2.4 Reducing the Workload

The designer can ease the grammar and the whole design by choosing *independent* bits of the underlying protocol to become part of the cover protocol, i.e. bits which can be set without taking other bits into account.

However, if we only take the bits we can set to "1" into account, it is possible to ease the formal grammar as well by decreasing the number of terminals ($a...c$ represent $a_1...c_1$). Certainly, this is only possible if there are no conditions for zero bits (e.g. "bit c is invalid, if bit a is zero"). The previously discussed type-2 grammar can be reduced to the grammar $G_{CP} = (V, \sum, P, S), V = \{S, A, B, C\}, \sum = \{a, b, c\}$:

$$P = \{S \rightarrow AB|AC \tag{10}$$

[1] If B is a regular language, $\overline{B}$ will be a regular language as well since the complement is closed within the regular languages.

[2] If both languages would be regular languages, their intersection would be a regular language. Thus, the emptiness problem would be decidable as well.

$$A \rightarrow a|\epsilon \tag{11}$$
$$B \rightarrow b|\epsilon \tag{12}$$
$$C \rightarrow bc\} \tag{13}$$

The previous type-3 grammar can also be reduced as follows:

$$P = \{S \rightarrow aB|bC|\epsilon \tag{14}$$
$$B \rightarrow bC|\epsilon \tag{15}$$
$$C \rightarrow c|\epsilon\} \tag{16}$$

In this case, a sentence not containing the terminal t implies that t_0 is present (otherwise, t_1 is would be present). For instance, the sentence ab stands for $a_1 b_1 c_0$. Instead of testing for $\{ACK, \neg DATA, DIS\}$, it is now required to test the reduced combination $\{ACK, DIS\} = ac$ using the reduced production rules in the same way as done before with the non-reduced production rules.

2.5 Handling Connection-oriented Protocols

The discussed approach using a formal grammar does not cover the problem of connection-oriented underlying protocols. In case such an underlying protocol is used, special rules can apply where previous states or packets have to be taken into account. Even if there is only a limited number of such situations, and although it is up to the protocol designer to select underlying protocol bits in a way that the modeling will be as easy as possible, the protocol designer probably has to deal with the problem nevertheless.

Different means from protocol engineering can be applied in this situation (like Petri nets or, for complex scenarios, composed I/O automata [13]). However, to stay with the approach of using formal grammar, we can extend the scenario by taking the necessary bits of previous packets into account. $\sum$ can contain two sets, the relevant bits of the previous packet i_p and the bits of the next packet to be sent i_n. Since only a very limited number of bits of previous packets (e.g. the SYN flag or the RST flag of TCP) are required to be considered here, the size of $\sum$ and P will not increase much.

For instance, if we assume a new packet can only set the bit b_n in case the previously sent packet contained the bit a_{ps} set or the previously received packet contained the bit z_{pr} set. The production rule for this scenario is easy to define: $X \rightarrow a_{ps}|z_{pr}$ and $B \rightarrow Xb_n|\epsilon$.

This solutions raises the questions of how to handle connection establishments where it is allowed to set a special bit (e.g. the SYN flag in TCP) that is not allowed to be set in packets other than the first one sent for each peer. Again, a solution is to define a new terminal that indicates such special situations (e.g. α in case no packet was sent and β in case a packet with a SYN flag was received). The resulting production rule is similar to the previous one: $X \rightarrow \alpha|\beta$ and $B \rightarrow Xb_n|\epsilon$.

As described by Lynch, it is feasible to model complex asynchronous systems by I/O automata composition, i.e. packet behaviour can be modeled in the context of a previous packet's behaviour. Yet, discussing automata composition is out of the scope of this paper and covered in detail by [13].

2.6 Iterative Design

The discussed 6 steps can be repeated until a solution fulfilling the requirements is found (e.g. more bits of the cover protocol are required to place the micro protocol within the provided cover protocol space).

To solve the above conflict of not being able to set the bits a and c without setting the bit b, it is necessary to relocate the mapping of the micro protocol list L_{MP} to the cover protocol list L_{CP} (step 5), which results in a less stealthy communication (if no mapping to equivalent values is possible). If this is not possible, i.e. no suitable mappings can be found, the micro protocol's functionality must be reduced (step 3) or the amount of available bits in the cover protocol must be increased (step 1). As mentioned in [23], it is thinkable to combine the functionality of multiple layers to gain enough cover protocol space.

Two solutions for the previous example can be created by simple modifications:

First solution: After a cover protocol re-definition, a fourth bit "d" could be discovered. Like "c", "d" is only only valid, if "b" is set. As mentioned earlier, we can define $G_{CP} = (V, \sum, P, S), V = \{S, A, B, C, D\}, \sum = \{a, b, c, d\}$ without the zero bit values:

$$P = \{S \rightarrow AB|AC \tag{17}$$
$$A \rightarrow a|\epsilon \tag{18}$$
$$B \rightarrow b|\epsilon \tag{19}$$
$$C \rightarrow bD|\epsilon \tag{20}$$
$$D \rightarrow c|d|cd\} \tag{21}$$

If we now map DATA to "d", we can always set "b" in case "c", "d", or "cd" is required. Thus, "abc" represents {ACK,DIS} under the conditions of the underlying protocol.

Second solution: By slightly reducing the functionality of the micro protocol, we can find a solution that does not require additional cover protocol bits.
a) The designer decides not to allow the combination {ACK,DIS}. Thus, the sender is required to send two separate packets: One packet for acknowledging data and a second packet to initiate a disconnect.
b) The mapping is switched, so that the ACK flag can only be set, if the DATA flag was set. In that case, the receiver must be capable of checking the presence of valid payload data. Thus, if {DATA, ACK} is set, DATA can be considered "invalid" if no payload data is present (e.g. all payload bits are zero). However, this solution depends on the use case of the covert communication and thus is out of the scope of our modelling.

As these example solutions reveal, our 6-step approach enables the protocol designer to switch back to any state of the design process at any time: If the validation is invalid, the mapping of L_{CP} and L_{MP}, the cover protocol (or parts of it), as well as the micro protocol header (or parts of it) can be re-designed. If on the other hand, the mapping results in a conflict ($sizeof(M) > sizeof(C)$), the micro protocol or the cover protocol can be re-designed. Fig. 1 visualizes the whole process and the possible re-engineering paths.

3 Results

As mentioned in Sect. 2.2, our two-layer approach aims to provide a simple solution for micro protocol engineering. Therefore, we ensure standard-conform behavior of the underlying network protocol by applying a formal grammar since it is a deterministic process to define the cover protocol as well as the micro protocol in this manner (cf. Sect. 2.3).

We additionally achieve to minimize the attention raised by the micro protocol by optimizing the mapping of bits between the underlying protocol and the micro protocol. As the selection process for network protocol areas and the optimized protocol switching (in case multiple network protocols are used simultaneously) was already discussed in [23] it is not part of this paper. Another advantage of our approach is to enable the protocol designer to switch between the 6 steps at any time as well as to enable the designer to apply the approach to any binary header protocol.

Using formal grammar becomes more complex if previous packet's values are required to be taken into account, as it can be the case for connection-oriented protocols. Also, there is no perfect way to model incremental values (like sequence numbers) and fields with many possible values (e.g. ICMP types in case all bits of the field are used). For these cases, we propose to either use a pre-defined list of possible values or not to take the field into account. However, integer fields with freely selectable values (like identifier fields) are easy to model using the production $X \rightarrow A_x ... N_x$ with $A_x \rightarrow a_0|a_1, ..., N_x \rightarrow n_0|n_1$.

A drawback of the presented two-layer approach is related to plaintext protocols, since the approach is designed to work with binary header protocols only. The grammar representation of underlying protocol rules is not practically adaptable to plaintext protocols – imagine a HTTP-based covert storage channel via different values for the HTTP "User-Agent", where many browsers, browser versions, architectures, and so on, can be used to signal hidden information. A production rule for such a scenario is only feasible if many grammar terminals are defined. However, using many terminals can result in a more complicated and error-prone grammar.

4 Conclusions

This paper presents a novel approach to design and verify covert channel-internal control protocols (so called micro protocols) ready for an implementation. The

approach is adaptable to different network protocols and can be applied in a systematic manner, typically with restricted effort. We argue that the attention raised by the micro protocol can be minimized by mapping the micro protocol's header bits to the header bits of the cover (and with it, the underlying) protocol in a way that the behavior of both protocols is as similar as possible. Using a simple example protocol, we demonstrate how to apply our approach. Due to the limited size of the micro protocol as well as given specifications, the grammar verification typically is a quick process. Also, our approach enables a dynamic re-designing and optimization based on its incremental design.

References

1. Baldoni, M., Baroglio, C., Martelli, A., Patti, V., Schifanella, C.: Verifying protocol conformance for logic-based communicating agents. In: Proc. 5th Int. Workshop on Comp. Logic in Multi-Agent Systems, CLIMA. pp. 192–212. Springer (2005)
2. Bauer, M.: New covert channels in HTTP: adding unwitting web browsers to anonymity sets. In: Proceedings of the 2003 ACM workshop on Privacy in the electronic society. pp. 72–78. WPES '03, ACM, New York, NY, USA (2003)
3. Bouajjani, A., Esparza, J., Finkel, A., Maler, O., Rossmanith, P., Willems, B., Wolper, P.: An efficient automata approach to some problems on context-free grammars. Inf. Process. Lett. 74(5-6), 221–227 (Jun 2000)
4. Giffin, J., Greenstadt, R., Litwack, P., Tibbetts, R.: Covert messaging through TCP timestamps. In: Proc. 2nd international conference on privacy enhancing technologies. pp. 194–208. Springer-Verlag, Berlin, Heidelberg (2003)
5. Girling, C.G.: Covert channels in LAN's. IEEE Transactions on Software Engineering 13, 292–296 (February 1987)
6. Gorodetski, V., Kotenko, I.: Attacks against computer network: Formal grammar-based framework and simulation tool. In: Recent Advances in Intrusion Detection, LNCS, vol. 2516, pp. 219–238. Springer Berlin / Heidelberg (2002)
7. Handley, M., Paxson, V., Kreibich, C.: Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In: 10th USENIX Security Symposium. vol. 10, pp. 115–131 (2001)
8. Koenig, H.: Protocol Engineering. Teubner (2003), (in German)
9. Lampson, B.W.: A note on the confinement problem. Commun. ACM 16(10), 613–615 (1973)
10. Li, W., He, G.: Towards a protocol for autonomic covert communication. In: Proceedings of the 8th international conference on autonomic and trusted computing. pp. 106–117. ATC'11, Springer-Verlag, Berlin, Heidelberg (2011)
11. Linn, R.J., McCoy, W.H.: Producing tests for implementations of OSI protocols. In: Protocol Specification, Testing, and Verification. pp. 505–520 (1983)
12. Lucena, N., Lewandowski, G., Chapin, S.: Covert channels in IPv6. In: Privacy Enhancing Technologies, LNCS, vol. 3856, pp. 147–166. Springer (2006)
13. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
14. McHugh, J.: Covert channel analysis. technical memo 5540:080a (1995)
15. Murdoch, S.J.: Covert channel vulnerabilities in anonymity systems. Ph.D. thesis, University of Cambridge (Computer Laboratory) (2007)
16. OpenBSD: pf.conf - packet filter configuration file (manual page) (July 2011)
17. Ray, B., Mishra, S.: A protocol for building secure and reliable covert channel. In: Korba, L., Marsh, S., Safavi-Naini, R. (eds.) PST. pp. 246–253. IEEE (2008)

18. Rowland, C.H.: Covert channels in the TCP/IP protocol suite. First Monday 2(5) (May 1997), http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/528/449, access: 2012-03-02
19. Rozenberg, G., Salomaa, A.: The Mathematical Theory of L Systems. Academic Press (1980)
20. Snort Project: Snort users manual 2.9.0 (March 2011)
21. Stødle, D.: Ping tunnel – for those times when everything else is blocked (2009), http://www.cs.uit.no/~daniels/PingTunnel/, access: 2012-03-05
22. Wendzel, S.: The problem of traffic normalization within a covert channel's network environment learning phase. In: Sicherheit'12. LNI, vol. 195, pp. 149–161 (2012)
23. Wendzel, S., Keller, J.: Low-attention forwarding for mobile network covert channels. In: De Decker, B., Lapon, J., Naessens, V., Uhl, A. (eds.) Communications and Multimedia Security. LNCS, vol. 7025, pp. 122–133. Springer (2011)
24. Yarochkin, F.V., Dai, S.Y., et al.: Towards adaptive covert communication system. In: PRDC. pp. 153–159. IEEE Computer Society (2008)
25. Zander, S., Armitage, G., Branch, P.: Covert channels and countermeasures in computer network protocols. IEEE Comm. Magazine 45(12), 136–142 (Dec 2007)