Spring 2022

Passing Time and Syncing Secrets: Demonstrating Covert Channel Vulnerabilities in Precision Time Protocol (PTP)

Aron J. Smith-Donovan
Macalester College, aronsmithdonovan@gmail.com

Macalester College — Saint Paul, MN

Department of Mathematics, Statistics, and Computer Science

Honors Thesis, Spring 2022

Passing Time and Syncing Secrets: Demonstrating Covert Channel Vulnerabilities in Precision Time Protocol (PTP)

completed by: Aron Smith-Donovan

advised by: Prof. Dr. Abigail Marsh

honors committee members: Prof. Dr. Elizabeth Shoop
Prof. Paul Cantrell

Abstract

Covert channels use steganographic approaches to transfer secret digital communications; when applied to network protocols, these strategies can facilitate undetectable data exfiltration and insertion attacks. Because covert channel techniques are protocol- and implementation-specific, individual case studies are necessary to assess for vulnerabilities under different conditions. While several investigations have been published evaluating covert channel potential in infrastructure- and manufacturing-based contexts, no existing research explores Precision Time Protocol (PTP), a time synchronization protocol commonly used in industrial control systems. This study aims to fill this gap by demonstrating the feasibility of a covert channel-based attack on a PTP-enabled network.

Table of contents

page #

List of figures

page #

List of tables

page #

1 Introduction

The term 'covert channels' generally refers to a set of strategies which conceal the existence of a digital information transfer. First defined as channels "not intended for information transfer at all" [1] and alternatively as channels that "use entities not normally viewed as data objects to transfer information" [2], they are distinct from *legitimate* channels which make use of a system's expected data mechanisms; the National Computer Security Center (NCSC) would later describe them as "communication channel[s] that [allow] a process to transfer information in a manner that violates the system's security policy" [3]. The key to these maneuvers is a straightforward one, though it can prove difficult to implement: a system cannot audit or intervene in communications that it does not know are occurring.

In a simple environment, such channels enable the breaking down of access control structures as processes with different security classifications communicate unimpeded; in a different setting, however, network covert channels between machines enable unrestricted, untraceable traffic. This increased scope brings in additional implications of data transfer at a far wider range, offering a potential avenue for existing threats—from information leaks to viral infections—to become even harder to trace.

The success of a given covert channel technique is protocol- and implementation-specific, meaning that both attack and remediation strategies require in-depth analyses of the context they are intended for [4]. Therein lies our predicament: covert channel attacks are laborious enough to pull off that assailants are likely to attempt an easier strategy whenever possible; however, in the event of a real covert channel attack, not only would countermeasures be slow to develop, but without targeted detection measures, it is possible the intrusion would pass by entirely unnoticed.

1.1 Investigation overview

The primary goal of this report is to determine the risk posed by covert channels to PTP-enabled networks. We begin with an overview of related works and the current state of research in section 2, including a targeted discussion of existing classification schemes in section 2.3. This is followed by an examination of Precision Time Protocol (PTP), covering use cases, protocol structure, and operation, in section 3.

In section 4, we conduct a systematic, theoretical evaluation of each pattern defined by the taxonomy chosen in section 2.3. This evaluation considers a given strategy's likelihood of success in a PTP-based message exchange, before classifying according to the potential throughput, detectability, and robustness of the channel. Of the sixteen approaches assessed, we found eleven to be possible, three of which we determined to be feasible.

Following the theoretical investigation in section 4, we developed a prototype of the strategy identified as the most auspicious; this implementation is described in section 5. We then performed a practical evaluation of the channel, with a summary of results provided in section 5.3 and our subsequent analysis in section 5.4. In the test environment, our prototype achieved a high throughput of seven bytes per message (bpm), averaging 24.36 bytes per second (bps) overall, as well as relatively low detectability, causing neither explicit errors nor blatant indicators of manipulation.

Given the results from our practical testing, we confirm that PTP-enabled networks are vulnerable to covert channel-based attacks. Our comprehensive risk assessment, as well as considerations for future work, are presented in section 6; the formal conclusion follows in section 7.

2 Related works

The study of covert channels is a long-standing, but niche, endeavor, with many attempts made at formal definitions and frameworks; presently, there exists a veritable buffet of proposed methods for defining, categorizing, evaluating, and comparing covert channels, with no single set of industry standards yet emerging. This section chronicles these historical developments alongside the current state of the field; we ultimately identify a single classification scheme best-suited for use in the present investigation and the imperative gap in existing research that this study aims to fill.

2.1 Early history of field

Butler Lampson's 1973 paper, "A note on the confinement problem", is widely considered to contain the first known use of the phrase 'covert channel' [1]. In the time since, the notion of a steganographic payload exploit has percolated into a wide range of research concerns, with numerous studies conducted to better characterize the issue. In 1983, the US DoD released the first edition of a standard called the Trusted Computer System Evaluation Criteria (TCSEC) [5], commonly called the "Orange Book", to establish requirements for assessing system security controls. This publication not only referred to Lampson's definition and recommended targeted analysis of covert channels, it divided relevant techniques into two categories still used today: storage channels, which communicate by modifying stored values, and timing channels, which communicate by modifying time intervals (both implicitly and explicitly).

In November 1993, a paper entitled "A Guide to Understanding Covert Channel Analysis of Trusted Systems" was put out by the same office in an effort to clarify the previously-provided covert channel recommendations [3]. This text collected existing definitions and research and made a concerted effort to precisely describe, classify, and develop evaluation metrics for covert channels; although this proposal informed many future endeavors, it did not establish a common standard.

2.2 Continued developments

The problem with delineating what a covert channel precisely *is* comes from the inalienable nature of the concept—they are fundamentally a negation, a set of communication methods that fall outside of what we agree to be legitimate [6]. Rather than pursue an explicit, comprehensive definition, many related projects have instead put forth enormous efforts to propose and catalog examples of covert channels as they are discovered through focused analyses. An early study by C.G. Girling explored the potential for a channel using packet delays between LAN-connected devices, serving as a precedent for later work to simply attempt covert communications and judge their level of success in doing so [7]. This methodology is capable of producing an alternate implicit definition of covert channels, illustrated by a corpus of examples.

The multitude of protocol-specific investigations in the last decade and a half have continued this general structure, from higher-level, text-heavy cases like Internet messaging protocols [8] and HTTP [9] to mid-level transport cases like TCP [10] and IPsec [11]. Performing these case-by-case analyses, however, has not eliminated the problem of a general definition for covert channels, but merely transformed it: having hashed out the practical bounds of what a covert channel can be, there remains the task of linking a wide assortment of research under some common understanding [12]. The underlying dilemma is one of relation: how can we consider these reports in conversation with each other? Given a growing pool of case studies, prior endeavors towards comparative models have been revisited with improved results, as addressed in section 2.3. This refinement of evaluation metrics for covert channels is a major focus of present research, and may ultimately aid the development of a universal definition.

2.3 Classification schemes

Myriad surveys to compile related covert channel techniques have been published as case studies continue to be conducted, including a noteworthy paper by Zander et al. which sorted many known attack and remediation strategies for network covert channels into a finite set of categories [13]. A string of publications from Wendzel et al. has endeavored not only to classify approaches, but to systematize them [14]; this renewed undertaking worked to develop a generalized model of covert channels by first refining categories into 'patterns' differentiated by the encoding approach used, the type of data object being affected (reminiscent of the initial storage/timing distinction [5]), and the possible usage contexts. The crucial addendum that makes this sorting system worthy of specific recognition is the sorting of the categories themselves: the patterns were arranged into a hierarchical taxonomy to allow for comparison between classifications at different levels of granularity as well as the option to continue to expand the pattern set in the face of new discoveries.

Effectively proving the elasticity of their model, Wendzel et al. went on to publish expansions and edits to the pattern set as part of a collection on systems security [15] and as a standalone paper entitled "A Revised Taxonomy of Steganography Embedding Patterns" [16]. This latest addition also included changes to further abstract existing patterns such that they could be applied to other steganographic domains, and, in a radical move towards open access, the current model has been made freely available online[1] [17]. The contents of this taxonomy are explored in detail in section 4, with each pattern considered for its potential in a PTP-targeted attack.

Wendzel et al.'s Information Hiding Patterns Project (IHPP) [12] furthermore has posited a description and evaluation structure for consideration as a new industry standard[2] [18]. Most current research makes use of three primary assessment criteria: capacity or throughput, which considers the amount of data able to be transmitted by a channel, detectability, which considers the extent to which the channel resists both generic and targeted detection measures, and robustness, which considers the extent to which the channel can communicate despite possible disruptions (e.g. network firewalls) [9, 11, 8, 19, 20, 21, 22]. The proposed extension to these criteria is acknowledged for its thoroughness

[1] https://patterns.ztt.hs-worms.de/

[2] https://patterns.ztt.hs-worms.de/desrcovert/

and potential for standardizing the field of study; however, within the scope of this paper, only the simplified set of three metrics will be explicitly named in the comparison between covert channel techniques, with the remainder of the proposal from Wendzel et al. used to inform the discussion.

2.4 Control systems

Following recent cases of infrastructure-focused cyberattacks, there has been an increased emphasis on understanding and preventing potential vulnerabilities of industrial control systems [23]. Such controllers are typically more straightforward than modern CPUs, utilizing simplified logic to manage the operation and automation of industrial processes; distributed control systems are used to coordinate processes at a larger scale, with more sophisticated machines managing the controllers from centralized locations [24]. Both post-incident investigations and controlled research have demonstrated that such controllers can be compromised to the point of physical damage with relatively small viral scripts [23].

In discussion with covert channels, control systems face a multi-directional risk: exfiltration of data from secure processes enables malicious actors to collect key operating information to direct other attacks, and code insertion into secure processes enables malicious actors to take down systems and potentially cause permanent damage to the equipment. For protocols in use below the application level, there are relatively few case studies for controller-specific protocols in comparison to case studies for Internet-specific and IP-based contexts. The more eminent of these controller-based projects include a study examining vulnerabilities in building automation protocols [19] and another looking at industrial control systems and the potential for supply chain attacks [21]; other lower-level, non-IP research includes a look at MQTT, the ping protocol for IoT devices [20], and a compelling prototype for a covert channel-based keylogger device [25].

The intention of the current investigation is, in part, an effort to continue the project of analyzing these controller protocols and documenting their demonstrable vulnerabilities. Because covert channels are difficult to detect without dedicated measures, it is difficult to say whether they have been used in real-world attacks, and there are typically easier methods available to a malicious actor [26]; in any case, control system attacks are an area of great concern with enormous potential for material consequences, and it is necessary to engage in these studies as a precautionary measure in anticipation of the point where the collected data becomes a crucial tool. This paper aims to contribute to this data by addressing the prospect of covert channels within Precision Time Protocol (PTP), a time synchronization protocol commonly implemented in control systems whose security is, therefore, of paramount importance; given the chronic under-documentation of covert channels in control systems, it is both alarming and unsurprising that there is not yet any publication devoted to this particular aim. An in-depth discussion of PTP's structure and operation can be found in section 3 of the present report, with the consideration of covert channels beginning in section 4.

3 PTP fundamentals

Precision Time Protocol (PTP) is a network protocol used to synchronize the internal clocks of locally-connected computer systems. First officially defined in 2002, it was designed as an alternative to an Internet-based time synchronization protocol called Network Time Protocol (NTP) as well as other GPS-based tools [27]. PTP is able to achieve an extremely high level of accuracy, reaching sub-nanosecond distinctions when properly configured, and was specifically designed for use on local area networks (LANs) where the majority of the linked systems are unable to communicate with an external time source, i.e. lacking the hardware for satellite or Internet connectivity [28, 29]. Instead of these external sources, devices synchronize to the internal clocks of other 'benchmark' devices on the same local network; this organization is discussed further in section 3.1.

PTP is a ubiquitous protocol, incorporated into a wide variety of use cases ranging from broadcast media systems [30] to IEEE's Audio Video Bridging (AVB) standards [31] to multiple IEC profiles for industrial automation [32, 33]; Wikipedia's list of PTP implementations has 138 entries as of January 2022 [34]. PTP is also being considered for future projects, including a proposal for use in Wide Area Monitoring (WAM) for power systems [35] and ongoing research by the Institute of Embedded Systems (InES) at the ZHAW [36].

Providing crucial functionality, the majority of PTP implementations are incorporated into other, more expansive frameworks; as such, standalone versions are less common. For this project, we used the open-source implementation `linuxptp` [37] due to its availability as a self-contained application of the PTP standard [28]. The following discussions of PTP's technical operation and structure are primarily based on and utilize examples from `linuxptp`.

3.1 Network architecture

In an established PTP network, devices that cannot access an external time source set their clocks based on one or more standard 'leader' clocks that are capable of such access, thereby serving as a standard time source internal to the local network. When a clock is first enabled, it broadcasts a series of `ANNOUNCE` messages containing information about itself that is used to establish a branched hierarchy for synchronization; in the `linuxptp` implementation, the clock is considered to be in the `INITIALIZING` state during this time, followed by the `LISTENING` state once the structure has been determined. The organization itself is handled by the Best Master Clock Algorithm (BMCA); the general grouping is outlined in figure 1, however this algorithm and the establishment of clock priority fall outside the scope of this paper and will not be explored in-depth.

It must be noted at this point that the official, established terminology for PTP and other similar protocols refer to this as a 'master-slave' structure, with higher-priority devices being the 'masters' and the lower-priority devices being the 'slaves'. This is not only archaic, but fails to accurately describe the relationship in question [38]. Despite efforts towards change [39], this language is still widely used and often considered the industry standard, and is written directly into the source code of `linuxptp` [37]. To the extent possible, the higher-priority devices will henceforth be referred to as 'leaders' and the lower-priority devices as 'followers'.

For our purposes, the network architecture will be simplified to consider only two nodes, as displayed in figure 2. In this setup, the sole follower clock syncs itself to the sole leader clock, which, in turn, has access to an external time source. In order to successfully synchronize, the follower clock must calculate the difference between its current time setting and the time setting of the leader clock while considering the impact of transmission and propagation delays; this process is described in section 3.3.

Figure 1: Generalized depiction of the BMCA-managed clock hierarchy.

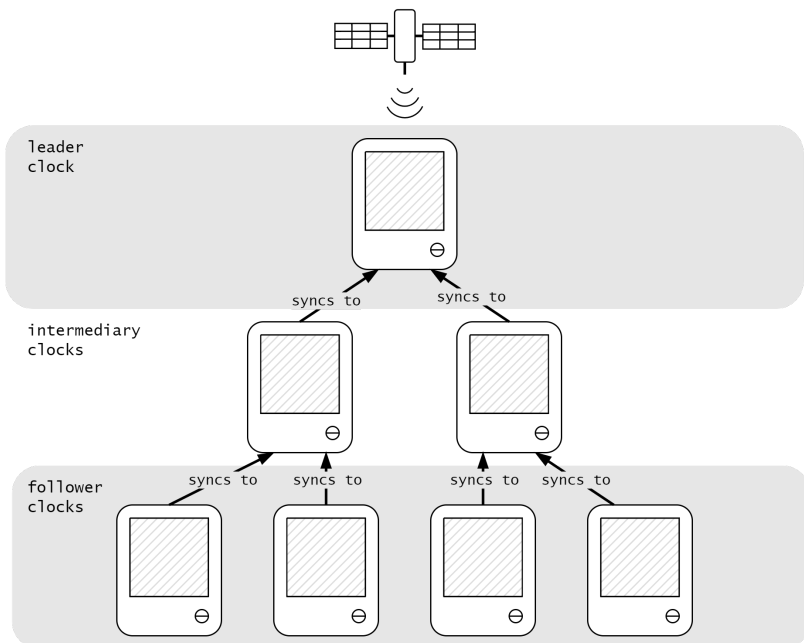
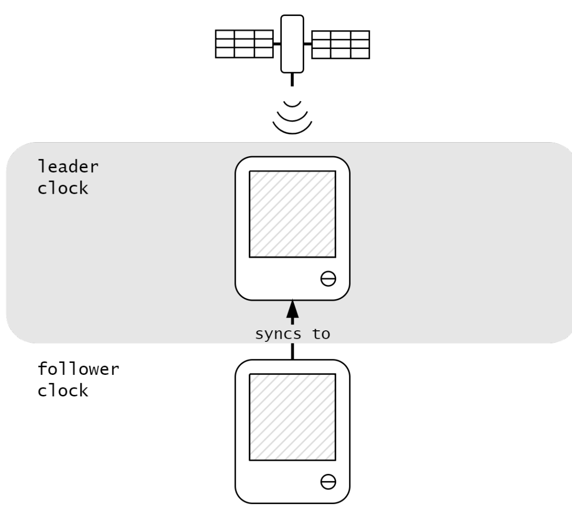


Figure 2: A simplified, two-node clock hierarchy.



3.2 Protocol structure

The following discusses a subset of the message types included in `linuxptp`, including their function and internal data structure. The message types not described here are the `MANAGEMENT` message, which is used to directly read and write clock parameters and typically disabled as a security practice, the `SIGNALING` message, which is used to transmit miscellaneous communications between clocks and is not part of standard operation, and the message variants specific to `linuxptp`'s peer-to-peer (P2P) mode; this study is only considering `linuxptp`'s default end-to-end (E2E) mode, and all future references to `linuxptp`'s behavior refer to this configuration unless explicitly specified otherwise. The information in this section has been sourced from the PTP standard [28] as well as directly from the `linuxptp` source and its associated documentation [37].

3.2.1 Header

Every `linuxptp` message includes a 34-byte message header; while the values written into its fields will vary, the structure is identical across message types.

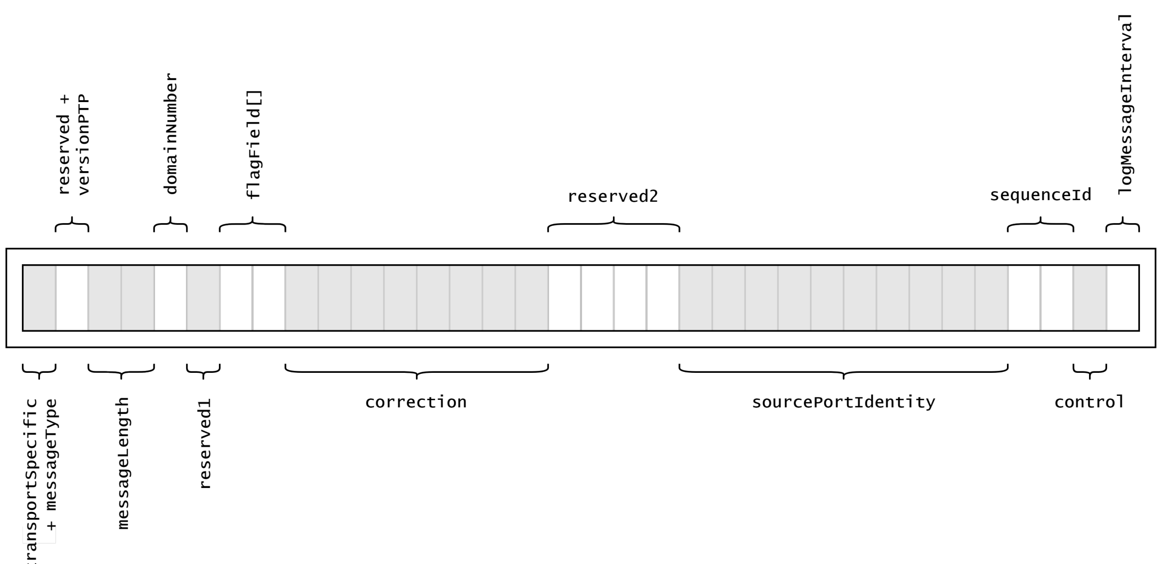


Figure 3: Diagram of a message header's byte structure; each block represents one byte, and the alternating white and grey sections indicate distinctions between fields.

Table 1: Descriptions of a message header's byte structure.

Field	Data type	Size	Description
`transportSpecific`	unsigned int	nibble	transport-specific field; shares a byte with `messageType`
`messageType`	unsigned int	nibble	current message type; shares a byte with `transportSpecific`
`reserved`	unsigned int	nibble	reserved field; shares a byte with `versionPTP`
`versionPTP`	unsigned int	nibble	current PTP version in use; shares a byte with `reserved`
`messageLength`	unsigned int	2 bytes	total length of message (includes header, body, and suffix)
`domainNumber`	unsigned int	byte	identifies the domain (= a logical grouping of clocks that synchronize to e/o with PTP) that the current message belongs to
`reserved1`	unsigned int	byte	reserved field
`flagField[]`	unsigned int	byte	an array to hold status flags
`correction`	int	8 bytes	correction value in nanoseconds for residence time within a transparent clock (= stateless, intermediary node that can exist between leader and follower clocks)
`reserved2`	unsigned int	4 bytes	reserved field
`sourcePortIdentity`	custom struct	10 bytes	the originating port for the current message
`sequenceId`	unsigned int	2 bytes	contains a sequence number for individual message types; used to link associated sets of `SYNC`, `FOLLOW_UP`, `DELAY_REQ`, and `DELAY_RESP` messages
`control`	unsigned int	byte	historical field; value depends on the message type; similar to the `messageType` field, but with fewer options
`logMessageInterval`	int	byte	message interval field; value depends on the message type

3.2.2 `ANNOUNCE` message

`ANNOUNCE` messages are continuously broadcast by all potential leader clocks to establish the synchronization hierarchy; BMCA dynamically decides the best organization based on the properties in this message. The default `linuxptp` behavior sends an `ANNOUNCE` message every 2 seconds, beginning when a clock successfully connects to the local network.

Figure 4: Diagram of an `ANNOUNCE` message's byte structure; each block represents one byte, and the alternating white and grey sections indicate distinctions between fields.

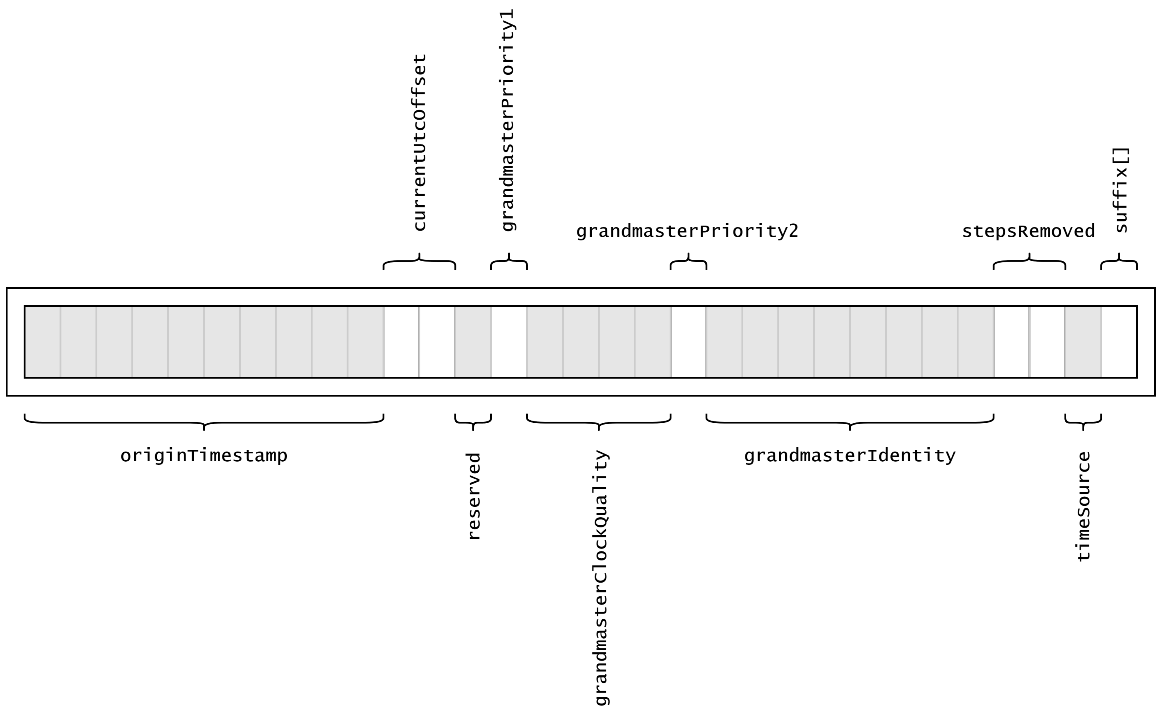


Table 2: Descriptions of an `ANNOUNCE` message's byte structure.

Field	Data type	Size	Description
`originTimestamp`	custom struct	10 bytes	approx. message timestamp at origin
`currentUtcOffset`	int	2 bytes	current UTC offset (timezone) at origin
`reserved`	unsigned int	1 byte	reserved field
`grandmasterPriority1`	unsigned int	1 byte	configurable clock priority (used for BMCA)
`grandmasterClockQuality`	custom struct	4 bytes	configurable clock quality (used for BMCA)
`grandmasterPriority2`	unsigned int	1 byte	configurable second-order clock priority (used for BMCA)
`grandmasterIdentity`	custom struct	8 bytes	configurable clock identity (used for BMCA)
`stepsRemoved`	unsigned int	2 bytes	network hop count
`timeSource`	unsigned int	1 byte	type of time source
`suffix[]`	unsigned int	1 byte	used to store TLV info

3.2.3 SYNC message

SYNC messages are sent from leader to follower to measure the clock offset in the L→F direction. Each message contains the approximate timestamp it was sent by the leader clock; the leader clock stores the exact local send timestamp after transmitting the message, and the follower clock stores the local timestamp that it received the message. The default linuxptp behavior sends a SYNC message every second, beginning when a leader clock successfully connects to the local network.

Figure 5: Diagram of a SYNC message's byte structure; each block represents one byte.

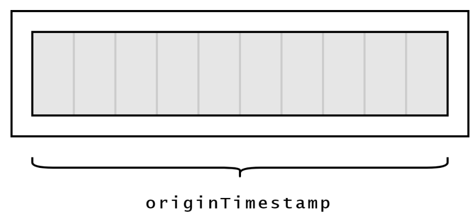


Table 3: Descriptions of a SYNC message's byte structure.

Field	Data type	Size	Description
originTimestamp	custom struct	10 bytes	approx. local send timestamp for current message

3.2.4 FOLLOW_UP message

FOLLOW_UP messages[3] are sent from leader to follower to relay the timestamp that an associated SYNC message was sent. Each message contains the exact local send timestamp recorded by the leader; the follower clock stores the timestamp contained in the message and does not record any other values. The default linuxptp behavior sends a FOLLOW_UP message every second at a <0.01 second delay from the preceding SYNC message.

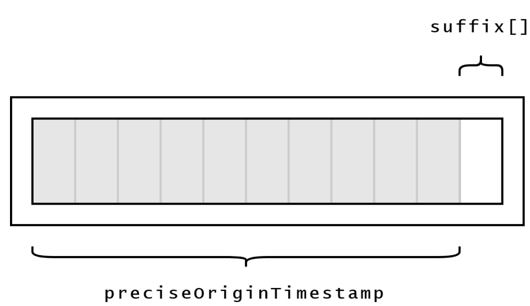


Figure 6: Diagram of a FOLLOW_UP message's byte structure; each block represents one byte, and the alternating white and grey sections indicate distinctions between fields.

[3]P2P variant: PDELAY_RESP_FUP

Table 4: Descriptions of a FOLLOW_UP message's byte structure.

Field	Data type	Size	Description
preciseOriginTimestamp	custom struct	10 bytes	exact local send timestamp for associated SYNC message
suffix[]	unsigned int	1 byte	used to store TLV info

3.2.5 DELAY_REQ message

DELAY_REQ (meaning 'delay request') messages[4] are sent from follower to leader to measure the clock offset in the F→L direction. Each message contains the approximate timestamp it was sent by the follower clock; the follower clock stores the precise local send timestamp after transmitting the message, and the leader clock stores the local timestamp that it received the message. The default linuxptp behavior sends a DELAY_REQ message approx. 10 seconds after receiving the associated FOLLOW_UP message.

Figure 7: Diagram of a DELAY_REQ message's byte structure; each block represents one byte, and the alternating white and grey sections indicate distinctions between fields.

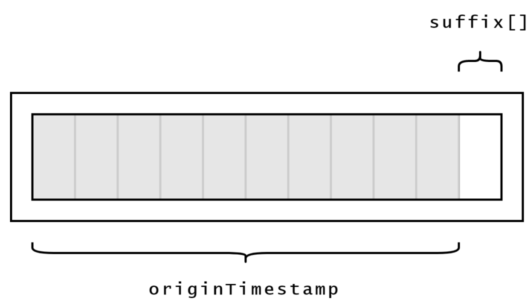


Table 5: Descriptions of a DELAY_REQ message's byte structure.

Field	Data type	Size	Description
originTimestamp	custom struct	10 bytes	approx. local send timestamp for current message
suffix[]	unsigned int	1 byte	used to store TLV info

3.2.6 DELAY_RESP message

DELAY_RESP (meaning 'delay response') messages[5] are sent from leader to follower to relay the timestamp that an associated DELAY_REQ message was received. Each message contains the exact local receipt timestamp recorded by the leader; the follower clock stores the timestamp contained in the message and does not record any other values. The default linuxptp behavior sends a DELAY_RESP message <1.0 second after receiving the associated DELAY_REQ message.

[4] P2P variant: PDELAY_REQ

[5] P2P variant: PDELAY_RESP

Figure 8: Diagram of a DELAY_RESP message's byte structure; each block represents one byte, and the alternating white and grey sections indicate distinctions between fields.

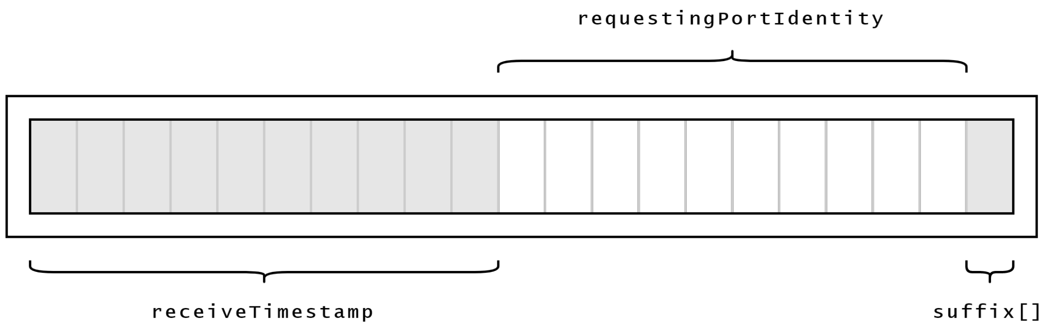


Table 6: Descriptions of a DELAY_RESP message's byte structure.

Field	Data type	Size	Description
receiveTimestamp	custom struct	10 bytes	exact local receive timestamp for associated DELAY_REQ message
requestingPortIdentity	custom struct	10 bytes	port identity of the clock that sent the associated DELAY_REQ message
suffix[]	unsigned int	1 byte	used to store TLV info

3.3 Synchronization procedure

To correct its clock setting, the follower clock must calculate the clock offset between itself and the leader clock while considering the impact of transmission and propagation delays; to do this, multiple messages are transmitted between the clocks and their send and receipt time recorded to provide the data needed for such a calculation, as shown in figure 9. This procedure can be described as follows:

1. A SYNC message is sent from leader to follower to measure the clock offset in the L→F direction; the SYNC message contains the approximate timestamp it was sent by the leader clock.
   (a) The leader clock stores the exact local send timestamp t_1 after transmitting the SYNC message.
   (b) The follower clock stores the local timestamp t_2 that it received the SYNC message.
2. A FOLLOW_UP message is sent from leader to follower to relay the timestamp that the associated SYNC message was sent; the FOLLOW_UP message contains the exact local send timestamp t_1 recorded by the leader.
   (a) The follower clock stores the timestamp t_1 contained in the FOLLOW_UP message.

3. A DELAY_REQ message is sent from follower to leader to measure the clock offset in the F→L direction; the DELAY_REQ message contains the approximate timestamp it was sent by the follower clock.

   (a) The follower clock stores the exact local send timestamp t_3 after transmitting the DELAY_REQ message.

   (b) The leader clock stores the local timestamp t_4 that it received the DELAY_REQ message.

4. A DELAY_RESP message is sent from leader to follower to relay the timestamp that the associated DELAY_REQ message was received; the DELAY_RESP message contains the exact local receipt timestamp t_4 recorded by the leader.

   (a) The follower clock stores the timestamp t_4 contained in the DELAY_RESP message.

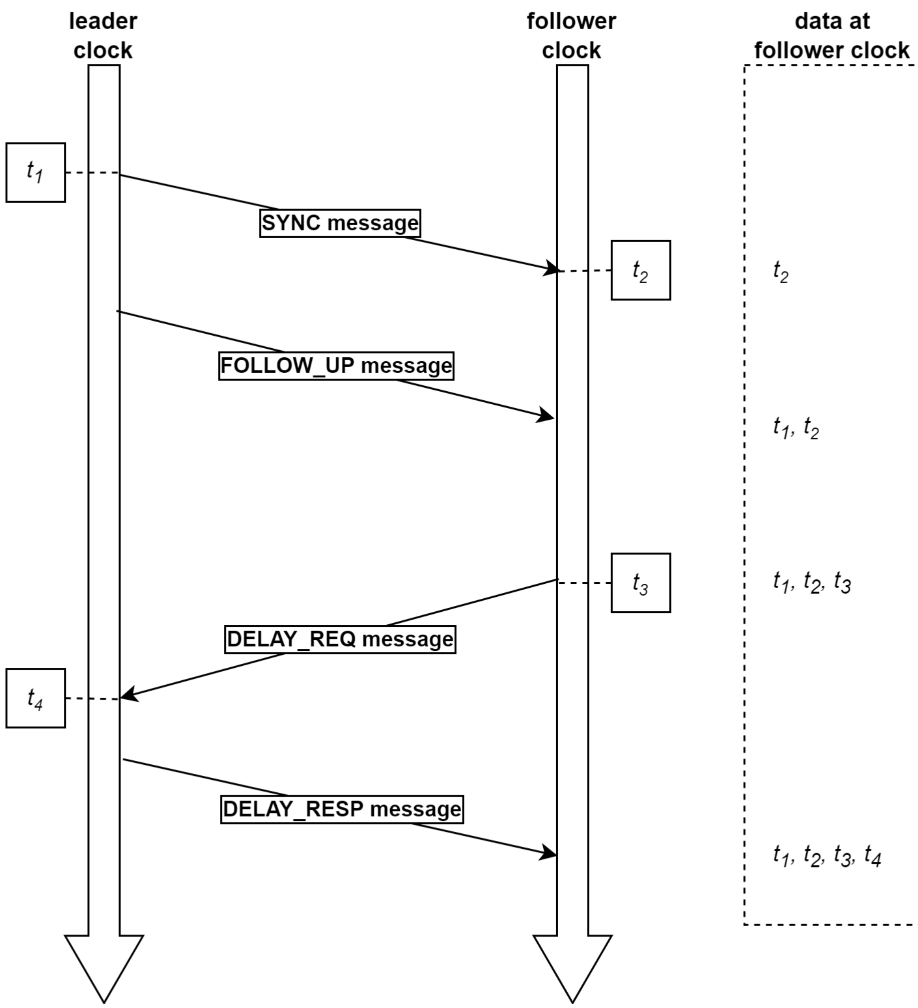


Figure 9: Simplified diagram of PTP's sync procedure.

After this message exchange, the follower clock has stored all four timestamp values (t_1, t_2, t_3, and t_4) and can calculate its offset. When a follower clock is first enabled, this exchange of messages is repeated to calculate multiple clock offsets with timestamps from distinct message sets; in the `linuxptp` implementation, the clock is considered to be in the `UNCALIBRATED` state during this time, with a default of 16 repetitions of the message exchange described in figure 9. At this point, the follower makes a large adjustment to its clock setting based on these offset values before entering the `SLAVE` state; in this state, message sets are continuously sent at a predetermined interval, with the follower clock calculating offset values to make small clock setting adjustments. The `SLAVE` state lasts until the connection to the leader clock is interrupted. During both the `UNCALIBRATED` and `SLAVE` states for the follower clock, the leader clock is considered to be in a `MASTER` state, where it remains until it loses connection to all of its follower clocks (causing it to return to the `LISTENING` state).

4 Exploration

For the theoretical phase of this study, each of the network covert channel patterns cataloged and taxonomized by Wendzel et al. [17] were considered for their feasibility when applied to a PTP-enabled setup. Potential channels were assessed by the amount of data they are able to transmit (throughput), the extent to which they resist both generic and targeted detection measures (detectability), and the extent to which they can communicate despite potential disruptions (robustness). See section 2.3 for additional explanation of these criteria.

The potential evaluation outcomes for each approach are ‘not possible’, designating approaches that lack suitable target objects within PTP’s structure or operation, ‘possible, but not feasible’, designating approaches that have suitable target objects but could not establish covert communications for a meaningful duration (e.g. approaches that cause terminal errors), and ‘feasible’, designating approaches that are capable of establishing covert communications.

4.1 PTP-based covert channels

4.1.1 Event/Element Interval Modulation - ET1 (RT1n)

Pattern definition. Messages are encoded by modulating the gaps between events/elements, e.g. modulating the inter-packet gap.

Assessment. PTP depends on consistent, symmetrical packet intervals and message delays to perform the message exchanges needed to successfully calculate clock offsets and synchronize connected devices; as such, modulating the intervals between elements is very likely to impair function, and the explicit errors this would cause render the channel highly detectable.

Evaluation. Possible, but not feasible.

4.1.2 Rate/Throughput Modulation - ET1.1 (RT1.1n)

Pattern definition. Messages are encoded by modulating the rate of events/elements; i.e., the message is not embedded into particular inter-event/element timings but in the overall rate/throughput.

Assessment. PTP messages are transmitted at set intervals defined explicitly in the source code, which thereby determines the overall transmission rates. These encoded interval values are needed to successfully calculate clock offsets and synchronize connected devices; as such, any modulation of message rates without manually adjusting the fields from within the source is very likely to impair function, and the explicit errors this would cause render the channel highly detectable.

Evaluation. Possible, but not feasible.

4.1.3 Event Occurrence - ET2 (RT2n)

Pattern definition. Messages are encoded in the temporal location of events; i.e., the rate of events is not directly modulated, but events are triggered at specific moments in time.

Assessment. PTP does not include any events that can be triggered at a specific point in time (with the exception of the connection/disconnection of devices, which falls under pattern RN1.1n); as such, this channel has no potential throughput and is not considered possible in a PTP context.

Evaluation. Not possible.

4.1.4 Frame Corruption - RT2.1n

Pattern definition. Messages are encoded by causing artificial frame collisions to signal the information.

Assessment. The potential throughput of this channel is highly restricted, as causing frame collisions at high rates is very likely to impair function, and the explicit errors this would cause render the channel highly detectable.

Evaluation. Possible, but not feasible.

4.1.5 Artificial Element-Loss - EN1 (RN1n)

Pattern definition. Messages are encoded by modulating the artificial loss of elements, e.g. dropping messages with an even sequence number.

Assessment. Sets of messages must contain all of their components in order to successfully calculate clock offsets and synchronize connected devices; such messages are linked together by the `sequenceId` field in the header, and while the artificial loss of an entire exchange could be performed, we know from the sync procedure that this still results in incomplete data. As such, the artificial loss of elements (whether individuals or groups) is very likely to impair function, and the explicit errors this would cause render the channel highly detectable.

Evaluation. Possible, but not feasible.

4.1.6 Artificial (Forced) Reconnections Modulation - RN1.1n

Pattern definition. Messages are encoded by causing artificial (forced) reconnections to signal the information; the covert sender influences connections of third-party nodes in a way that their connections to either a central element (e.g., an MQTT broker or a server) or a peer (in a peer-to-peer network) are terminated and then established again (i.e., a reconnect is performed).

Assessment. PTP-enabled clocks can be manually disconnected and reconnected, prompting them to observably re-enter the `INITIALIZING` state. The initial synchronization period after a clock connects to the network causes a delay that restricts the throughput for this channel; in addition, communication is likely to be unidirectional, as disconnecting and reconnecting leader clocks would cause repeated shifts in the BMCA-managed clock hierarchy, thereby increasing the channel's detectability (i.e. this channel is far more feasible for communications originating at follower clocks). In general, this channel is not likely to be detectable without targeted countermeasures, as disconnections are a normal part of PTP operation and will not cause any explicit errors; however, depending on the broader implementation, errors may be thrown by other monitoring policies.

Evaluation. Feasible.

4.1.7 Elements/Features Positioning - EN2 (RN2n)

Pattern definition. Messages are encoded by modulating the position of a predefined (set of) element(s)/feature(s) in a sequence of elements/features, e.g. changing the position of an IPv4 option in the list of options.

Assessment. PTP does not include any sequences of elements with configurable positions; as such, this channel has no potential throughput and is not considered possible in a PTP context.

Evaluation. Not possible.

4.1.8 Elements/Features Enumeration - EN3 (RN3n)

Pattern definition. Messages are encoded by altering the overall number of appearances of elements/features in a sequence, e.g. fragmenting a network packet into a specific number of fragments, modulating the number of people wearing a t-shirt in a specific color in an image file.

Assessment. PTP does not include any sequences of elements with configurable numbers of appearances; as such, this channel has no potential throughput and is not considered possible in a PTP context.

Evaluation. Not possible.

4.1.9 Artificial Retransmissions Modulation - RN3.1n

Pattern definition. Messages are encoded by re-transmitting previously sent or received PDUs.

Assessment. Sending previously received messages is very likely to impair function because of the call-and-response message pattern, and the explicit errors this would cause render the channel highly detectable. PTP messages that are not used in calculations are already sent at regular intervals with no requirement to modify their contents and can be re-transmitted without error, e.g. `ANNOUNCE` messages and `MANAGEMENT` messages (when enabled). PTP messages used in calculations are linked together by the `sequenceId` field in the header and can also be re-transmitted without error, as the contents of these messages will only ever be used together with the contents from the other messages in the set. The throughput for this channel is restricted by the amount of new traffic that could be introduced into the network before being easily noticeable by non-targeted countermeasures.

Evaluation. Feasible.

4.1.10 State/Value Modulation - EN4 (RN4n)

Pattern definition. Messages are encoded by modulating the states or values of features, e.g. changing values of the network packet header fields.

Assessment. Because of the function that PTP provides, the vast majority of message contents are metadata pertaining to the clocks on the network; while it is unlikely that a person will view the contents of any given message, the contents still cannot readily be modified without causing rampant calculation errors. As such, overwriting values is very likely to impair function, and the explicit errors this would cause render the channel highly detectable.

Evaluation. Possible, but not feasible.

4.1.11 Reserved/Unused State/Value Modulation - EN4.1 (RN4.1n)

Pattern definition. Messages are encoded by modulating reserved/unused states/values, e.g. overwriting the IPv4 reserved field.

Assessment. PTP contains several ‘reserved’ fields in both the message header and in the body of certain message types, as well as fields intended for bitwise values that do not fill their allocated space; such fields can be overwritten without error. The throughput of this channel is restricted by the amount of unused space in a given message type and the established rate of message transmission. In addition, this channel may be detectable by non-targeted countermeasures (e.g. data validation) and is likely to be robust against most barriers (e.g. firewalls) that do not directly modify message values, although it is not likely to be robust against scenarios involving data loss.

Evaluation. Feasible.

4.1.12 Random State/Value Modulation - EN4.2 (RN4.2n)

Pattern definition. Messages are encoded by replacing a (pseudo-)random value or (pseudo-) random state with a secret message (that is also following a pseudo-random appearance), e.g. replacing the pseudo-random content of a network header field with encrypted covert content.

Assessment. PTP does not include any random or pseudo-random elements; as such, this channel has no potential throughput and is not considered possible in a PTP context.

Evaluation. Not possible.

4.1.13 Blind State/Value Modulation - EN4.3 (RN4.3n)

Pattern definition. Messages are encoded by blindly corrupting of data, e.g. blindly overwriting a checksum to either corrupt a packet or not.

Assessment. Because of the function that PTP provides, the vast majority of message contents are metadata pertaining to the clocks on the network; while it is unlikely that a person will view the contents of any given message, the contents still cannot readily be modified without causing rampant calculation errors. As such, blind overwriting is very likely to impair function, and the explicit errors this would cause render the channel highly detectable.

Evaluation. Possible, but not feasible.

4.1.14 Feature Structure Modulation - EN5 (RN5n)

Pattern definition. Comprises all hiding techniques that encode messages by modulating the structural properties of a feature (but not states/values (EN4), positions (EN2) or number of appearances (EN3)), e.g. increasing/decreasing the size of succeeding network packets.

Assessment. PTP message sizes are defined explicitly in the source code. These encoded size metrics are used to extract the values needed to successfully calculate clock offsets and synchronize connected devices; as such, any modulation of structural properties without manually adjusting the fields from within the source is very likely to impair function, and the explicit errors this would cause render the channel highly detectable.

Evaluation. Possible, but not feasible.

4.1.15 Size Feature Modulation - EN5.1 (N5.1n)

Pattern definition. Messages are encoded by modulating the size of an element, e.g. create additional (unused) space in network packets for embedding hidden data.

Assessment. All PTP elements are of a set size defined explicitly in the source code. Theses encoded size metrics are used to extract the values needed to successfully calculate clock offsets and synchronize connected devices; as such, any modulation of feature sizes without manually adjusting the fields from within the source is very likely to impair function, and the explicit errors this would cause render the channel highly detectable.

Evaluation. Possible, but not feasible.

4.1.16 Character Feature Modulation - EN5.2 (RRN5.2n)

Pattern definition. Messages are encoded by modulating different features in characters, such as color, size (scale), font, position or size of different parts in some letters, e.g. using upper- or lowercase letters.

Assessment. PTP does not include any text-based elements; as such, this channel has no potential throughput and is not considered possible in a PTP context.

Evaluation. Not possible.

4.2 Summary of findings

Pattern name	Evaluation	Reason
Event/Element Interval Modulation	possible, but not feasible	likely to impair function and/or throw errors → high detectability
Rate/Throughput Modulation	possible, but not feasible	likely to impair function and/or throw errors → high detectability
Event Occurrence	not possible	no suitable target objects
Frame Corruption	possible, but not feasible	viable throughput conditions likely to impair function and/or throw errors → high detectability
Artificial Element-Loss	possible, but not feasible	likely to impair function and/or throw errors → high detectability
Artificial (Forced) Reconnections Modulation	feasible	not likely to throw errors; throughput limited by network configuration; false-positive rate high
Elements/Features Positioning	not possible	no suitable target objects
Elements/Features Enumeration	not possible	no suitable target objects
Artificial Retransmissions Modulation	feasible	not likely to throw errors; increased throughput conditions also increase detectability
State/Value Modulation	possible, but not feasible	likely to impair function and/or throw errors → high detectability
Reserved/Unused State/Value Modulation	feasible	not likely to throw errors; throughput limited by established message rates; false-positive rate low
Random State/Value Modulation	not possible	no suitable target objects
Blind State/Value Modulation	possible, but not feasible	likely to impair function and/or throw errors → high detectability
Feature Structure Modulation	possible, but not feasible	likely to impair function and/or throw errors → high detectability
Size Feature Modulation	possible, but not feasible	likely to impair function and/or throw errors → high detectability
Character Feature Modulation	not possible	no suitable target objects

5 Experimentation

Given the three patterns identified as feasible in section 4, 'Reserved/Unused State/ Value Modulation' has the greatest likelihood of success as a vehicle for a covert channel attack of a PTP-enabled network due to its high throughput potential and low propensity to cause errors or false-positives. This approach is hereafter interchangeably referred to as 'unused value modulation' and 'unused value'. This section discusses the development of a prototype implementation for this pattern and its experimental evaluation; section 6 will cover these practical findings in conversation with those from section 4.

5.1 Methodology

5.1.1 Technical specifications

The unused value modulation channel was realized using a fork of the open-source PTP implementation `linuxptp`[6] [37] on a network testbed comprised of two NVIDIA® Jetson Nano™ 2GB machines[7] communicating over Ethernet. Both devices are identically equipped with 128-core NVIDIA Maxwell™ GPUs, Quad-core ARM® Cortex-A57 CPUs, 2GB 64-bit LPDDR4 SDRAM, 64GB microSD internal storage, and 802.11ac wireless Internet connectivity; in addition, both were configured with Ubuntu 18.04.6 LTS from a device image provided by NVIDIA[8] in a headless setup, with root access from the external computer over Ethernet. These test machines were able to successfully synchronize their clocks with `linuxptp`, and all recorded traffic is genuine; for both legitimate and encoded messages, data was captured using the open-source, packet-monitoring software Wireshark[9] v2.6.10, with filters configured to log only PTP messages.

5.1.2 Determining channel targets

The `linuxptp` message structures contain several reserved fields, as described in section 3.2, but naming conventions alone cannot reliably authenticate whether a given field is unused. To confirm potential targets for the chosen covert channel, we assessed each message field to determine if its value is written and/or read during standard operation; table 8 includes the evaluation of each field along with a brief justification. Through this analysis, we identify six unused fields, one of which is in the body of the `ANNOUNCE` message structure, with the remaining five found in the message header. While `ANNOUNCE` messages are sent at regular intervals during standard operation, their `reserved` field provides one byte of capacity per message where the header fields collectively provide seven. In the interest of the present prototype, the channel targets consist of only the unused fields located in the message header.

[6] https://github.com/aronsmithdonovan/linuxptp-CC
[7] https://developer.nvidia.com/embedded/jetson-nano-2gb-developer-kit
[8] https://developer.nvidia.com/jetson-nano-2gb-sd-card-image
[9] https://www.wireshark.org/

Table 8: Evaluation of message fields as targets for the unused value covert channel.

Parent structure	Field name	Size	Unused?	← How do we know?
message header	`transportSpecific`	nibble	NO	field consistently empty on initial observation; attempted arbitrarily overwriting value and clock sync failed
message header	`messageType`	nibble	NO	value is often referenced for use in switch statements and crucial to operation; only uses the decimal values 0, 1, 2, 3, 8, 9, 10, 11, 12, 13, but because it is always read as a decimal value, there are no unused bits that data could be written into
message header	`reserved`	nibble	YES	field consistently empty on initial observation; attempted arbitrarily overwriting value and clock sync succeeded without errors
message header	`versionPTP`	nibble	NO	field is crucial to operation and hard-set to 2 in multiple files, therefore not suitable
message header	`messageLength`	2 bytes	NO	field is crucial to operation and referenced for use in calculations, therefore not suitable
message header	`domainNumber`	1 byte	NO	clock domains are completely separated from each other and function as distinct virtual networks; modifying this field prevents clock sync from succeeding, therefore not suitable
message header	`reserved1`	1 byte	YES	field consistently empty on initial observation; attempted arbitrarily overwriting value and clock sync succeeded without errors
message header	`flagField[0]`	1 byte (nibble unused)	YES	field contains flags that trigger behavioral changes when set, however only the last three bits are used; attempted arbitrarily assigning a value to the first five bits and clock sync succeeded without errors
message header	`flagField[1]`	1 byte	NO	field contains flags that trigger behavioral changes when set, however only the last seven bits are used; attempted arbitrarily assigning a value to the first bit and clock sync succeeded without errors, but because of its exceedingly low capacity, this field is removed from consideration
message header	`correction`	8 bytes	NO	field is crucial to operation and referenced for use in calculations, therefore not suitable

Continued on next page.

Parent structure	Field name	Size	Unused?	← How do we know?
message header	`reserved2`	4 bytes	YES	field consistently empty on initial observation; attempted arbitrarily overwriting value and clock sync succeeded without errors
message header	`sourcePortIdentity`	10 bytes	NO	field is crucial to operation, therefore not suitable
message header	`sequenceId`	2 bytes	NO	field is crucial to operation and referenced for use in calculations, therefore not suitable
message header	`control`	1 byte	YES	in the `linuxptp` documentation, this is described as a redundant field for `messageType`; its possible values are enumerated in the `msg.h` file, but the field is not read anywhere in the source; attempted arbitrarily overwriting value and clock sync succeeded without errors
message header	`logMessageInterval`	1 byte	NO	field handles message intervals and is crucial for operation, therefore not suitable
`ANNOUNCE`	`originTimestamp`	10 bytes	NO	field is crucial to operation and referenced for use in calculations, therefore not suitable
`ANNOUNCE`	`currentUtcOffset`	2 bytes	NO	field is crucial to operation and referenced for use in calculations, therefore not suitable; always read as a decimal value, there are no unused bits that data could be written into
`ANNOUNCE`	`reserved`	1 byte	YES	field consistently empty on initial observation; attempted arbitrarily overwriting value and clock sync succeeded without errors
`ANNOUNCE`	`grandmasterPriority1`	1 byte	NO	field is crucial to operation and required for BMCA calculations, therefore not suitable
`ANNOUNCE`	`grandmasterClockQuality`	4 bytes	NO	field is crucial to operation and required for BMCA calculations, therefore not suitable
`ANNOUNCE`	`grandmasterPriority2`	1 byte	NO	field is crucial to operation and required for BMCA calculations, therefore not suitable
`ANNOUNCE`	`grandmasterIdentity`	8 bytes	NO	field is crucial to operation and required for BMCA calculations, therefore not suitable
`ANNOUNCE`	`stepsRemoved`	2 bytes	NO	field is referenced for use in calculations, therefore not suitable

Continued on next page.

Parent structure	Field name	Size	Unused?	← How do we know?
ANNOUNCE	timeSource	1 byte	NO	field is crucial to operation and referenced for use in calculations, therefore not suitable
ANNOUNCE	suffix[]	1 byte	NO	field is checked upon reception of every message and throws an error for any unexpected values, therefore not suitable
SYNC	originTimestamp	10 bytes	NO	field is crucial to operation and referenced for use in calculations, therefore not suitable
DELAY_REQ	originTimestamp	10 bytes	NO	field is crucial to operation and referenced for use in calculations, therefore not suitable
DELAY_REQ	suffix[]	1 byte	NO	field is checked upon reception of every message and throws an error for any unexpected values, therefore not suitable
FOLLOW_UP	preciseOriginTimestamp	10 bytes	NO	field is crucial to operation and referenced for use in calculations, therefore not suitable
FOLLOW_UP	suffix[]	1 byte	NO	field is checked upon reception of every message and throws an error for any unexpected values, therefore not suitable
DELAY_RESP	receiveTimestamp	10 bytes	NO	field is crucial to operation and referenced for use in calculations, therefore not suitable
DELAY_RESP	requestingPortIdentity	10 bytes	NO	field is crucial to operation, therefore not suitable
DELAY_RESP	suffix[]	1 byte	NO	field is checked upon reception of every message and throws an error for any unexpected values, therefore not suitable

End of table.

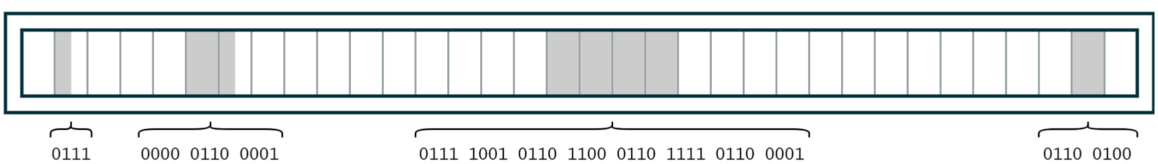
5.2 Implementation

To demonstrate the unused value channel in the message header for `linuxptp`, we chose to create a proof-of-concept implementation that hides a text payload in the target fields identified in section 5.1.2. This example is sufficient for the experimental evaluation and subsequent discussion of the channel, as the basic mechanism of writing and reading bitwise values in the header operates identically for other payload formats.

The chosen targets allow us to encode seven bytes in every PTP message transmitted by either clock type; crucially, this space is non-contiguous and accepts only numerical values. To comply with the latter restriction, each character of the text payload is encoded as its 8-bit ASCII value, but addressing the former requires some additional consideration: while the total throughput is a whole number of bytes, this includes two 4-bit segments, thereby necessitating the splitting of individual ASCII values across multiple fields.

Our final encoding method pulls seven characters of the payload text at a time, saves each of their ASCII values as two separate, 4-bit values, and inserts these pieces into the target fields as they fit. For example, the payload string `"payload"` contains seven characters and can be encoded into a single message: the intact hexadecimal ASCII values for these characters are {`0x70, 0x61, 0x79, 0x6C, 0x6F, 0x61, 0x64`} and the split 4-bit binary values are {`0111, 0000, 0110, 0001, 0111, 1001, 0110, 1100, 0110, 1111, 0110, 0001, 0110, 0100`}, with the allocation of this sample data shown in figure 10.

Figure 10: Example of data allocation to encode the string `"payload"`; each block represents one byte, and the grey sections indicate unused fields.



The described encoding scheme is implemented in the `hdr_pre_send` function within the `msg.c` file of our `linuxptp` fork.[10] In its unmodified state, `hdr_pre_send` performs final validation of the message header fields before transmission, and is called within the `msg_pre_send` function; this existing component was not modified or removed to construct the prototype channel. We have integrated the encoding directly into the source to better demonstrate the relationship between the unused `linuxptp` header fields and the actual encoding process, and a similar script to the one we have developed could serve the same function without access to the source code by referencing the physical locations rather than the variable names. The modified `hdr_pre_send` function containing the encoding implementation is provided in its entirety in section A.1.

Our decoding method performs these same steps in reverse, pulling the separated ASCII values out of the headers of received messages and reassembling them before writing the associated characters to a local file. This is implemented in the `hdr_post_recv` function within the `msg.c` file of our `linuxptp` fork.[11] Similarly to `hdr_pre_send`, in its unmodified state, `hdr_post_recv` performs initial processing of the message header fields immediately

[10] https://github.com/aronsmithdonovan/linuxptp-CC
[11] https://github.com/aronsmithdonovan/linuxptp-CC

after reception, and this component was not modified or removed to construct the prototype channel. The modified `hdr_post_recv` function containing the decoding script is provided in its entirety in section A.2.

5.3 Results

To evaluate the channel implementation described in section 5.2, we recorded both legitimate `linuxptp` traffic (i.e. without the covert channel enabled) and encoded `linuxptp` traffic (i.e. with the covert channel enabled) between our test machines with Wireshark, as described in section 5.1.1. Each sample was collected over a ten-minute period beginning with the initialization for both devices and including the entire sync procedure as described in section 3.3. For the encoded messages, the payload was transmitted in both directions, with each machine performing the encoding of sent messages and the decoding of received messages; throughput calculations for this case assume seven bytes per message.

5.3.1 Message type data

Table 9: Message type counts for legitimate PTP traffic, given over an approx. 612.83-second period.

Message type	Count/612.83 seconds
`ANNOUNCE`	307 messages
`SYNC`	612 messages
`FOLLOW_UP`	612 messages
`DELAY_REQ`	602 messages
`DELAY_RESP`	602 messages

Table 10: Message type counts for encoded PTP traffic, given over an approx. 607.52-second period.

Message type	Count/607.52 seconds
`ANNOUNCE`	304 messages
`SYNC`	607 messages
`FOLLOW_UP`	607 messages
`DELAY_REQ`	597 messages
`DELAY_RESP`	597 messages

5.3.2 Traffic data

Table 11: Results summary for legitimate PTP traffic; decimal values have been truncated.

	Value		
Metric	Legitimate leader	Legitimate follower	Legitimate overall
time connected	612.83 seconds	604.03 seconds	612.83 seconds
total transmitted	2133 messages	602 messages	2735 messages
transmission rate	3.48 messages/second	0.99 messages/second	4.46 messages/second

Table 12: Results summary for encoded PTP traffic; decimal values have been truncated.

	Value		
Metric	Encoded leader	Encoded follower	Encoded overall
time connected	607.52 seconds	600.81 seconds	607.52 seconds
total transmitted	2115 messages	597 messages	2712 messages
transmission rate	3.48 messages/second	0.99 messages/second	4.46 messages/second
channel throughput	24.36 bytes/second	6.95 bytes/second	24.36 bytes/second

5.4 Analysis

The message rate data immediately reinforces the asymmetrical activity implied by the sync procedure described in section 3.3, i.e. significantly more messages are sent by leader clocks than follower clocks, resulting in an uneven throughput. We conclude from this that leader clocks are at a higher risk in the event of an exfiltration attack than they are in the event of an insertion attack, and follower clocks are at a higher risk in the event of an insertion attack than they are in the event of an exfiltration attack. Despite this discrepancy between the clock types, our prototype channel exhibited a relatively high throughput in either direction, with approx. 24.3 bytes per second (bps) from the leader and approx. 6.9 bps from the follower.

Another significant finding from this data is the lack of a detectable change in message rates: the values for the legitimate and encoded cases are remarkably close, and are considered equivalent within a margin of error of ± 0.01 seconds. As discussed in section 3.3 and section 4.1, PTP depends on consistent message intervals to calculate clock offsets, and any behavior that modifies message timing is very likely to impair function. Our channel prototype did not cause explicit errors at any point during testing, and given the level of timing consistency, it is highly unlikely that a network controller would detect any change in transmission; as such, we conclude the channel to have a relatively low detectability with non-targeted countermeasures.

It is possible, however, that the unused values channel could be detectable with targeted remediation strategies. For example, the `linuxptp` implementation lacks data validation for its historical and reserved fields, allowing our implemented channel to pass by without errors; if data validation were present in a network monitor or within `linuxptp` itself, the channel would cause errors and therefore become highly detectable. An exhaustive review of countermeasure strategies falls outside the scope of this study, and potential future endeavors are described in section 6.2.

Due to equipment limitations, the robustness of this channel has not been thoroughly explored. It is worth noting at this point that the encoded messages passed through an Ethernet network switch without issue, and that the payload text was continuously, repeatedly transmitted for extended connected sessions without any observed error on the receiving end. Again, potential future investigations are discussed in section 6.2.

6 Discussion

The prototype described in section 5 confirms the feasibility of a covert channel attack targeting PTP-enabled networks. In testing, the unused value modulation approach displayed a bidirectional throughput of seven bytes per message (bpm), with approx. 24.3 bytes per second (bps) transmitted from leader clock to follower clock and approx. 6.9 bps transmitted from follower clock to leader clock; no behaviors detectable by untargeted measures were observed. It is possible that more complex PTP integrations could be capable of detecting the channel, however these cases warrant their own dedicated investigations and are, along with the development of targeted countermeasures, beyond the scope of this paper.

Where the implemented strategy stores covert communications directly in data objects, the two covert channel patterns determined to be feasible in section 4 that were not used for the prototype in section 5 are both indirect channels, meaning information is instead encoded in behavioral changes. If they were to be implemented, both approaches are predicted to have very high detectability due to this observable modification of network events, as well as a throughput of <1 bit per message.

Considering the ongoing difficulty of generalizing covert channel studies as described in section 2.2 and section 2.3, a broad claim of a given attack's performance would likely be unsubstantiated. Within the assessed `linuxptp` context, the unused value channel proved to be not only feasible, but highly successful. The implications of these findings are considered in section 6.1.

6.1 Risk assessment

As described in section 2.4, PTP is a common protocol in industrial and manufacturing settings using local network setups; the explicit requirement described in section 3.1 that at least one locally connected device have a second, external network connection provides a potential entry point for a malicious actor. Due to the access level required to establish a covert channel, they are generally more advantageous, and therefore more likely to appear, as data transfer tools within larger attack scenarios; their risk potential has been assessed accordingly.

Outgoing covert channels facilitate the exfiltration of data from a secure system to an insecure one; in a PTP setup, potentially vulnerable information can include specifications and schedules for controllers and equipment. While unauthorized access to such details may not be inherently harmful, knowledge of the intended target can inform future attacks; for example, device specifications can be used to develop malware, and equipment schedules are helpful in planning physical access or (D)DoS attacks. Incoming covert channels, on the other hand, introduce the potential for insertion- and injection-based exploits. PTP's hierarchical network structure, discussed in section 3.1, renders systems highly susceptible to self-propagating malware (worms), and simpler logic controllers are vulnerable to disruption by relatively small viral scripts [23].

The execution of a covert channel-based attack requires highly detailed knowledge of the target, presenting a level of difficulty that makes a malicious actor more likely to pursue an easier exploit strategy [13]. In addition, because these attacks are designed to be undetectable, real-world incidence data is nearly nonexistent—this scarcity is exacerbated by security clearance policies, which typically consider attempted attacks to be at a higher level than the targeted resource(s), further limiting the availability of this information [26]. Knowing this, any evaluation of risk can only be made in consideration of the potential damage if an attack were carried out, with the likelihood of such an attack still unidentified.

6.2 Future work

Given the dilemma of accurately predicting the risk of a covert channel attack discussed in section 6.1, it is often challenging to justify research on prevention measures. However, because covert channels can only reliably be detected with targeted remediation strategies [13], such investigations are vital to understanding the actual incidence rates of these attacks; it is ultimately inadvisable to delay securing systems until we are sure these weaknesses are actively being exploited.

Many potential directions for future work remain; continued study into the PTP vulnerabilities described in this report, as well as into developing, evaluating, and implementing countermeasures, is recommended at this point. Additional testing is needed on the robustness of the implemented `linuxptp` channel against non-ideal circumstances (e.g. firewalls). Lastly, other PTP implementations, both standalone and those incorporated into larger frameworks, require their own separate case studies in a similar convention to this paper's study of `linuxptp`.

6.3 Limitations

This study faced restrictions caused by available data and equipment capabilities. The body of existing work on covert channels is not cohesive, and the lack of established industry standards hinders the generalization of results; this is explored in section 2. There is relatively little research on covert channels in control systems and no existing studies on covert channels in PTP, resulting in a lack of clear precedent in the field. In addition, access to the source code for `linuxptp` allowed for a higher level of familiarity with the attack context than would be possible in most real-world scenarios, and the network testbed used consisted of only two devices, thereby not allowing for all scenarios possible with `linuxptp` to be evaluated: the peer-to-peer (P2P) mode of operation was not explored, and certain message types were not able to be produced, as discussed in section 3 and section 5.

7 Conclusion

Covert channel strategies are protocol- and implementation-specific, requiring individual case studies to assess for vulnerabilities. Through this paper, we have demonstrated the feasibility of covert channels in Precision Time Protocol (PTP). Of the feasible attack strategies identified, one was implemented and experimentally evaluated; the high throughput and low detectability of this prototype illustrate the need for additional work developing remediation strategies. This project is the first to assess PTP for covert channel vulnerabilities, and intends to contribute to the body of research on covert channels in control systems as well as prompt improvements to better secure PTP-enabled systems.

A Appendix

A.1 Payload encoding script

```c
size_t pos = 0;  // file position holder

/**
 * Processes the message header immediately prior to sending. Located in the msg.c file and called within the msg_pre_send function.
 *
 * @param m     The header structure of the message being prepared for transmission.
 */
static int hdr_pre_send(struct ptp_header *m)
{
  // INITIALIZATION
    char *filename = "payload.txt";  // payload source file
    unsigned int ch;  // holder for individual chars in payload text
    unsigned int payload[14];  // holder for numerical encodings of payload chars
    int i, j;  // iterators

  // OPEN PAYLOAD SOURCE FILE
    FILE *fp = fopen(filename, "r");

  // ERROR CHECK
    if(fp == NULL) {
      printf("Error: could not open file %s", filename);
    }

  // RETURN TO SAVED FILE POSITION
    fseek(fp, pos, SEEK_SET);

  // READ NEXT 7 CHARS OF PAYLOAD
    for(i = 0; i<14; i++) {  // for 14 repetitions...
      ch = fgetc(fp);  // get next char in payload text
      switch(ch == EOF) {
        case 0:  // if current char does not mark the end of the payload file...
          payload[i] = (unsigned int)(ch >> 4);  // save the first digit of the current char's ASCII value
          i++;  // move forward one spot in the holder array
          payload[i] = (unsigned int)(ch & 0x0f);  // save the second digit of the current char's ASCII value
          break;  // exit switch statement and continue to next char in payload text
        default:  // if current char does mark the end of the payload file...
          for(j=i; j<14; j++) {  // for the remainder of the space in the holder array...
            payload[j] = (unsigned int)(0x0);  // save the first digit of the ASCII value for new line
            j++;  // move forward one spot in the holder array
            payload[j] = (unsigned int)(0xa);  // save the second digit of the ASCII value for new line
          }
          goto reset_file;  // reset file position to zero
      }
    }

  // SAVE FILE POSITION
    pos = ftell(fp);
    reset_file_return:

  // CLOSE PAYLOAD FILE
    fclose(fp);

  // WRITE PAYLOAD TO HEADER FIELDS
```

```
    // reserved (nibble)
      m->ver = m->ver | (payload[0]<<4);
    // reserved1 (1 byte)
      m->reserved1 = (payload[1]<<4) | payload[2];
    // flagField[0] (nibble)
      m->flagField[0] = m->flagField[0] | (payload[3]<<4);
    // reserved2 (4 bytes)
      m->reserved2 = (payload[4] << 28) |
              (payload[5] << 24) |
              (payload[6] << 20) |
              (payload[7] << 16) |
              (payload[8] << 12) |
              (payload[9] << 8) |
              (payload[10] << 4) |
              payload[11];
    // control (1 byte)
      m->control = (payload[12] << 4) | payload[13];

  // CONVERT BYTE ORDER
  //// included in original linuxptp source; converts target values from host CPU byte order to network byte order
    m->messageLength = htons(m->messageLength);
    m->correction = host2net64(m->correction);
    m->sourcePortIdentity.portNumber = htons(m->sourcePortIdentity.portNumber);
    m->sequenceId = htons(m->sequenceId);

  // RETURN
  //// included in original linuxptp source
    return 0;

  // RESET FILE
    reset_file:
      pos=0;  // set file position to zero
      goto reset_file_return;  // complete the rest of the function

}
```

A.2 Payload decoding script

```
/**
 * Processes the message header immediately after receiving. Located in the msg.c
     ↪ file and called within the msg_post_recv function.
 *
 * @param m     The header structure of the message being processed after reception.
 */
static int hdr_post_recv(struct ptp_header *m)
{
  // CONVERT BYTE ORDER
    //// included in original linuxptp source; converts target values from network
     ↪ byte order to CPU byte order
    if ((m->ver & VERSION_MASK) != VERSION)
      return -EPROTO;
    m->messageLength = ntohs(m->messageLength);
    m->correction = net2host64(m->correction);
    m->sourcePortIdentity.portNumber = ntohs(m->sourcePortIdentity.portNumber);
    m->sequenceId = ntohs(m->sequenceId);

  // READ HEADER FIELDS AND WRITE PAYLOAD TO FILE
    FILE *exfp;
    exfp = fopen("exfiltrated-payload.txt", "a");
    fprintf(exfp, "%c", (m->ver & 0xf0) | (m->reserved1 >> 4));
    fprintf(exfp, "%c", ((m->reserved1 & 0x0f) << 4) | (m->flagField[0] >> 4));
    fprintf(exfp, "%c", (m->reserved2 >> 24) & 0xff);
    fprintf(exfp, "%c", (m->reserved2 >> 16) & 0xff);
    fprintf(exfp, "%c", (m->reserved2 >> 8) & 0xff);
    fprintf(exfp, "%c", (m->reserved2) & 0xff);
    fprintf(exfp, "%c", (m->control) & 0xff);
    fclose(exfp);

  // RETURN
    //// included in original linuxptp source
    return 0;
}
```

Bibliography

[1] Butler W Lampson. "A note on the confinement problem". In: *Communications of the ACM* 16.10 (1973), pp. 613–615.

[2] Richard A Kemmerer. "Shared resource matrix methodology: An approach to identifying storage and timing channels". In: *ACM Transactions on Computer Systems (TOCS)* 1.3 (1983), pp. 256–277.

[3] US Department of Defense. *Covert Channel Analysis of Trusted Systems (Light Pink Book)*. National Computer Security Center, Fort Meade, MD. 1993. URL: https://irp.fas.org/nsa/rainbow/tg030.htm.

[4] James P. Anderson. *Computer Security Technology Planning Study*. Planning study. Deputy for Command and Management Systems (Bedford, MA), 1972. URL: http://seclab.cs.ucdavis.edu/projects/history/papers/ande72.pdf.

[5] US Department of Defense. *Trusted Computer System Evaluation Criteria (Orange Book)*. National Computer Security Center (Fort Meade, MD). 1983. URL: https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/dod85.pdf.

[6] *Overt and covert channels*. Certified Ethical Hacker (CEH) Blog. Jan. 2012. URL: http://certifiedethicalhackerceh.blogspot.com/2012/01/overt-and-covert-channels.html.

[7] C.G. Girling. "Covert Channels in LAN's". In: *IEEE Transactions on Software Engineering* SE-13.2 (1987), pp. 292–296. DOI: 10.1109/TSE.1987.233153. URL: https://ieeexplore.ieee.org/document/1702208.

[8] Reshad Patuck and Julio Hernandez-Castro. "Steganography using the extensible messaging and presence protocol (XMPP)". In: *arXiv preprint arXiv:1310.0524* (2013).

[9] Erik Brown et al. "Covert channels in the HTTP network protocol: Channel characterization and detecting Man-in-the-Middle attacks". In: *Journal of Information Warfare* 9.3 (2010), pp. 26–38.

[10] Hanaa Nafea et al. "Efficient non-linear covert channel detection in TCP data streams". In: *IEEE Access* 8 (2019), pp. 1680–1690.

[11] Steffen Schulz, Vijay Varadharajan, and Ahmad-Reza Sadeghi. "The silence of the LANs: efficient leakage resilience for IPsec VPNs". In: *IEEE transactions on information forensics and security* 9.2 (2013), pp. 221–232.

[12] Steffen Wendzel et al. *About*. Information Hiding Patterns Project. June 2021. URL: https://patterns.ztt.hs-worms.de/about/.

[13] Sebastian Zander, Grenville Armitage, and Philip Branch. "A survey of covert channels and countermeasures in computer network protocols". In: *IEEE Communications Surveys & Tutorials* 9.3 (2007), pp. 44–57.

[14] Steffen Wendzel et al. "Pattern-based survey and categorization of network covert channel techniques". In: *ACM Computing Surveys (CSUR)* 47.3 (2015), pp. 1–26.

[15] Sebastian Zillien and Steffen Wendzel. "Reconnection-Based Covert Channels in Wireless Networks". In: *ICT Systems Security and Privacy Protection*. IFIP SEC 2021. Springer International Publishing, pp. 118–133.

[16] Steffen Wendzel et al. "A Revised Taxonomy of Steganography Embedding Patterns". In: *arXiv preprint arXiv:2106.08654* (2021).

[17] Steffen Wendzel et al. *Pattern collection*. Information Hiding Patterns Project. June 2021. URL: https://patterns.ztt.hs-worms.de/PatternCol/.

[18] Steffen Wendzel et al. *Describing covert channels*. Information Hiding Patterns Project. June 2021. URL: https://patterns.ztt.hs-worms.de/desrcovert/.

[19] Steffen Wendzel, Benjamin Kahler, and Thomas Rist. "Covert channels and their prevention in building automation protocols: A prototype exemplified using BACnet". In: *2012 IEEE International Conference on Green Computing and Communications*. IEEE. 2012, pp. 731–736.

[20] Aleksandar Velinov et al. "Covert channels in the MQTT-Based Internet of Things". In: *IEEE Access* 7 (2019), pp. 161899–161915.

[21] Mario Hildebrandt et al. "Information hiding in industrial control systems: An OPC UA based supply chain attack and its detection". In: *Proceedings of the 2020 ACM Workshop on Information Hiding and Multimedia Security*. 2020, pp. 115–120.

[22] Ryan Mazerik. *ICMP attacks*. INFOSEC Institute. Aug. 2021. URL: https://resources.infosecinstitute.com/topic/icmp-attacks/.

[23] Andy Greenberg. *Sandworm: A New Era of Cyberwar and the Hunt for the Kremlin's Most Dangerous Hackers Hardcover*. Doubleday, 2019. ISBN: 978-0385544405.

[24] *Industrial Control System*. Trend Micro. URL: https://www.trendmicro.com/vinfo/us/security/definition/industrial-control-system.

[25] Gaurav Shah, Andres Molina, Matt Blaze, et al. "Keyboards and Covert Channels." In: *USENIX Security Symposium*. Vol. 15. 2006, p. 64.

[26] Luke Muehlhauser. *Jonathan Millen on covert channel communication*. Machine Intelligence Research Institute. July 2014. URL: https://intelligence.org/2014/04/12/jonathan-millen/.

[27] John C. Eidson. *Measurement, Control and Communication Using IEEE 1588*. Springer, Apr. 2006. ISBN: 978-1-84628-250-8.

[28] *IEEE 1588-2019. Standard for a Precision Clock Synchronization Protocol for networked measurement and Control Systems*. IEEE Standards Association. Sept. 2016. URL: https://standards.ieee.org/ieee/1588/6825/.

[29] *Precision Time Protocol (PTP/IEEE-1588)*. White paper. EndRun Technologies. URL: https://endruntechnologies.com/pdf/PTP-1588.pdf.

[30] Aimée Ricca. "SMPTE Publishes First Two Parts of Standard Enabling Deployment of PTP-Timed Equipment in Existing SDI Plants". In: *Society of Motion Picture and Television Engineers* (Apr. 2015).

[31] Geoffrey M. Garner. "IEEE 802.1AS and IEEE 1588". In: Joint ITU-T/IEEE Workshop The Future of on The Future of Ethernet Transport. Geneva, Switzerland, May 2010. URL: https://www.itu.int/dms_pub/itu-t/oth/06/38/T06380000040002PDFE.pdf.

[32] *Industrial communication networks - High availability automation networks. Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR).* International Electrotechnical Commission. URL: https://webstore.iec.ch/publication/64423.

[33] *Communication networks and systems for power utility automation. Part 9-3: Precision time protocol profile for power utility automation.* International Electrotechnical Commission. URL: https://webstore.iec.ch/publication/24998.

[34] Wikipedia contributors. *List of PTP implementations.* URL: https://en.wikipedia.org/wiki/List_of_PTP_implementations (visited on 01/01/2022).

[35] Antonio Pepiciello and Alfredo Vaccaro. "A reliable architecture based on Precision Time Protocol for WAMPAC synchronization". In: *2018 AEIT International Annual Conference*. 2018, pp. 1–5. DOI: 10.23919/AEIT.2018.8577414.

[36] *Institute of Embedded Systems (InES).* URL: https://www.zhaw.ch/en/engineering/institutes-centres/ines/.

[37] Richard Cochran et al. *linuxptp. PTP IEEE 1588 stack for Linux.* Latest commit f078f19339a3 51be8bccd1daed59c4845918d30d. GitHub repository. Aug. 2020. URL: https://github.com/openil/linuxptp.

[38] Peter Kirn. *Let's dump master-slave terms: They're vague, horrible, and we're better off without them.* Create Digital Media (CDM). June 2020. URL: https://cdm.link/2020/06/lets-dump-master-slave-terms/.

[39] Elizabeth Landau. *Tech confronts its use of the labels 'master' and 'slave'.* WIRED. July 2020. URL: https://www.wired.com/story/tech-confronts-use-labels-master-slave/.