A Formal Analysis of SCTP: Attack Synthesis and Patch Verification

Jacob Ginesin*
ginesin.j@northeastern.edu
Northeastern University

Max von Hippel*
vonhippel.m@northeastern.edu
Northeastern University

Evan Defloor†
defloor.e@northeastern.edu
Northeastern University

Cristina Nita-Rotaru†
c.nitarotaru@northeastern.edu
Northeastern University

Michael Tüxen†
tuexen@fh-muenster.de
FH Münster

Abstract

SCTP is a transport protocol offering features such as multi-homing, multi-streaming, and message-oriented delivery. Its two main implementations were subjected to conformance tests using the PACKETDRILL tool. Conformance testing is not exhaustive and a recent vulnerability (CVE-2021-3772) showed SCTP is not immune to attacks. Changes addressing the vulnerability were implemented, but the question remains whether other flaws might persist in the protocol design.

We study the security of the SCTP design, taking a rigorous approach rooted in formal methods. We create a formal PROMELA model of SCTP, and define 10 properties capturing the essential protocol functionality based on its RFC specification and consultation with the lead RFC author. Then we show using the SPIN model checker that our model satisfies these properties. We next define 4 representative attacker models – Off-Path, where the attacker is an outsider that can spoof the port and IP of a peer; Evil-Server, where the attacker is a malicious peer; Replay, where an attacker can capture and replay, but not modify, packets; and On-Path, where the attacker controls the channel between peers. SCTP was designed to be secure against Off-Path attackers, and we study the additional models in order to understand how its security degrades for successively more powerful attacker types. We modify an attack synthesis tool designed for transport protocols, KORG, to support our SCTP model and 4 attacker models.

We synthesize the vulnerability reported in CVE-2021-3772 in the Off-Path attacker model, when the patch is disabled, and we show that when enabled, the patch eliminates the vulnerability. We also manually identify two ambiguities in the RFC, and using KORG, we show that each, if misinterpreted, opens the protocol to a new Off-Path attack. We show that SCTP is vulnerable to a variety of attacks when it is misused in the Evil-Server, Replay, or On-Path attacker models (for which it was not designed). We discuss these and, when possible, mitigations thereof. Finally, we propose two RFC errata – one to eliminate each ambiguity – of which so far, the SCTP RFC committee has accepted one.

*Contributed equally.
†Listed alphabetically.

1 Introduction

Transport protocols play a crucial role in transmitting data across the Internet either directly – as in UDP [3] and DCCP [22], which provide unreliable communication, and TCP [20] and SCTP [58], which provide reliable communication – or by supporting secure protocols – e.g., UDP supports DTLS [51] and QUIC [31], while TCP supports TLS [50]. Thus, it is critical that transport protocols are designed and implemented to be bug-free and secure.

SCTP is a transport layer protocol proposed as an alternative to TCP, offering new features, such as multi-homing, multi-streaming, and message-oriented delivery. Among other use-cases, it is the data channel for WebRTC [5], which is used by such applications as Facebook Messenger [35], Microsoft Teams [37], and Discord [65]. The design of SCTP is described in RFC documents, the most recent one being RFC 9260 [58], and implemented in Linux [2] and FreeBSD [4]. These implementations were tested using PACKETDRILL [1, 14] and analyzed with WIRESHARK [52]. Some limited efforts also analyzed the SCTP design using formal methods. The works in [62, 63] focused only on bugs and did not consider attacks, while the work in [53] focused on attacks, but modeled only limited aspects of connection establishment to compare the resilience of SCTP and TCP to SYN-FLOOD attacks. A recent vulnerability – CVE-2021-3772 [49] – shows the importance of conducting a much more comprehensive formal analysis. Although a patch was proposed in RFC 9260 [58], and adapted by FreeBSD, the question remains whether other flaws might persist in the protocol design and whether the patch might have introduced additional vulnerabilities. To the best of our knowledge, no prior works formally analyzed the entire SCTP connection establishment and teardown routines in a security context.

In this work, we take an approach rooted in formal methods to study the security of SCTP. Our approach is based on *attack*

synthesis, where the goal is, given a program that behaves correctly, and an attacker model, to find an attack that can lead the program to behave incorrectly.[1] Specifically, we use an attack synthesis tool called KORG [67] which can find attacks where an attacker manipulates two protocol peers in order to induce a denial of service (DoS). Combined with other formal methods, such as model checking, this approach allows us to precisely study the behaviors of SCTP under different DoS attacker models.

Model Design and Verification. We start by creating finite state machine (FSM) models for the consecutive SCTP designs specified in RFCs 4960 [55] and 9260 [58], and writing ten properties the models should satisfy based on a close reading of the RFC documents and discussions with the lead SCTP author [2]. Our properties are defined in Linear Temporal Logic (LTL) and characterize the standard establishment and teardown routines, the proper functioning of the cookie timer, and the fact that SCTP does not support half-open connections. Using the SPIN model checker, we automatically verify that our SCTP model meets these properties (behaves correctly) when not under attack.

Attack Synthesis. We use an attack synthesis tool for transport protocols called KORG, based on LTL model-checking [67]. KORG automatically finds DoS attacks against the protocol's handshake (establishment and teardown) in which an attacker, constrained to scenarios with two peers, sends and receives messages in order to guide the peers into a deleterious state or loop. Note that KORG does not find other types of attacks, such as statistical attacks against pseudo-random values used in the protocol or side-channel attacks against protocol implementations, and is limited to aspects of the protocol (such as its handshake) which are inherently finite-state. Consequently, KORG cannot be used to study the commonly used Dolev-Yao attacker model [19], in which the attacker is allowed to launch an unbounded number of simultaneous connections, making it the wrong choice to study cryptographic protocols such as TLS. This limitation is acceptable because the Dolev-Yao attacker model is inappropriate for studying DoS attacks against transport protocol handshakes, as we do in this paper.[3] KORG is based on LTL model-checking, thus it suffers from state-space explosion and so does not scale to systems with large state-spaces. However, the handshake mechanisms of transport protocols such as TCP, DCCP, and SCTP are small, and their correctness criteria can be written in LTL, so KORG is an appropriate choice for this context.

We define four attacker models (Off-Path, Evil-Server, Replay, and On-Path), which are representative for transport protocols and provide a wide range of attacker capabilities allowing us to understand the behavior of SCTP when under DoS attack by an attacker who cannot open additional connections parallel to the one under attack. The Off-Path attacker model describes an attacker who may or may not know the IP address or port of either peer, but cannot read the messages in-transit, and does not know the authentication secrets (which in SCTP are called the "vtags") of the association. Thus, its injected messages should theoretically be ignored. In the Evil-Server attacker model, one peer in an association is malicious, and aims to guide the other peer into some vulnerable state. The Replay attacker model describes an attacker capable of capturing messages from the communication channel and replaying them without modification. In the On-Path attacker model, the attacker controls the channel connecting the peers, and can intercept, drop, and inject authenticated messages at-will. Note that SCTP was designed to withstand Off-Path attackers but was not designed to be fully robust against the other three attacker models. Thus, we study the Evil-Server, Replay, and On-Path attacker models only to understand what could happen in a "worst case scenario".

Using KORG, we automatically synthesize DoS attacks against our SCTP model, for each LTL property and attacker model. In the Off-Path case, we automatically find the attack from CVE-2021-3772. We find numerous DoS attacks in the Evil-Server and On-Path attacker models, e.g., an Evil-Server attack that establishes a connection with the victim peer and then leaves it stranded, and an On-Path attack that injects messages guiding both peers into Shutdown_Received (an illegal passive/passive teardown). These results highlight the importance of implementation level defenses against an Evil-Server, and an end-to-end security model to prevent On-Path attacks. We also find one Replay attack, highlighting the criticality of the transmission sequence number (TSN).

Patch Verification. We next configure the model to include the patch introduced in RFC 9260 [58] and show that the patch fixes the problem, i.e., the LTL property that was violated by the attack is now met under the attacker model where that attack was discovered. We further show that of the ten LTL properties we use to specify the correctness of SCTP, none which were satisfied before the patch are violated after it is introduced, in any of the four attacker models we study. KORG is sound and complete, meaning, (1) if it finds an attack then the attack is real (against the model), and (2) if any attacks against the model and property exist, of the type KORG looks for, then given sufficient time and memory, KORG will find one [67]. If KORG runs out of time or memory without finding an attack, it will report "Search not completed". While this might happen for larger state machines, it did not happen for the SCTP model and any of the properties or attacker models we studied in this paper. Thus, our analysis suffices to show

[1] This is totally different from program synthesis, where the problem is, given some property, to conjure a program that satisfies it.

[2] We do not seek to construct a complete set of properties, as we're interested in studying the security-relevant behaviors of SCTP rather than creating an all-encompassing specification. Also, defining a complete specification in LTL is impractical, as LTL is optimized for efficient model checking.

[3] The Dolev-Yao attacker, by definition, controls the communication channels and therefore can pull off a DoS trivially. Therefore, any DoS attack discovered using Dolev-Yao must be manually analyzed to confirm that it is non-trivial.

that the patch does not enable any vulnerability against any property which was previously upheld, from the list of ten properties we specify, in any of the four attacker models we study. However, it does not carry any implications for other kinds of attacks, such as statistical attacks against the itag/vtag scheme or side-channel attacks against implementations; nor does it imply anything about properties or attacker models other than the ones we studied. It is therefore possible that the patch could introduce an attack outside the scope of what we analyzed using KORG.

RFC Disambiguation. Motivated by the fact that CVE-2021-3772 was caused by a lack of clarity in RFC 4960, we carefully manually analyze its replacement, RFC 9260, for ambiguities. We identify two portions of RFC 9260 that seem ambiguous, and show that each can be interpreted in two ways. We confirm the correct interpretation for each by consulting with the lead SCTP RFC author; then model the incorrect interpretation and synthesize attacks against it. We propose RFC errata to clarify the correct interpretation of each. Finally, we use PACKETDRILL [14] to confirm that the Linux and FreeBSD implementations interpret the ambiguous portions correctly. Note, the FreeBSD implementation was co-authored by the lead SCTP RFC author, so naturally it interprets the RFC correctly.

Contributions. We summarize our contributions:

- *Model:* We model the original SCTP RFC [55] using PROMELA. Our model can be configured with or without the CVE patch from RFC 9260 [58]. It is endorsed by the lead SCTP RFC author and faithfully captures the SCTP connection and teardown routines, including the exchange of messages, the user-on-the-loop and its commands, and the handling out-of-the-blue packets.
- *Verification:* We formalize ten novel correctness properties for SCTP in LTL based on a close reading of the RFCs and use SPIN to prove that our model satisfies all ten when no attacker is present.
- *Attack Synthesis:* We introduce four attacker models for SCTP. Then we modify KORG to support *packets* and *replay attacks*, and use it to synthesize DoS attacks in the context of each. For Off-Path, we rediscover the CVE before the patch was applied, but not after. For Evil-Server, we find four attacks that, depending on implementation details, could leave a victim peer deadlocked or stranded in some liveness cycle, unable to automatically de-associate. For Replay, we find one attack that, depending on the security of the TSN, could prevent two peers from establishing a connection. We find six similar On-Path attacks where the attacker leads the peers into some illegal state or cycle, violating a property.
- *Patch Verification:* We show that the patch fixes the problem, i.e. the property that was violated by the attack is now met under the attacker model wherein the CVE attack was discovered. Moreover, we show that in each attacker model, the patch does not open the protocol to any vulnerabilities of the kind KORG looks for against properties which were not previously vulnerable to attack.
- *RFC Disambiguation:* We identify two ambiguities in RFC 9260, each of which, we show, could be reasonably misinterpreted in a way that opens the protocol to a new vulnerability. We confirm that neither implementation makes either mistake, and to avoid mistakes in future implementations, we suggest two RFC errata clarifying the ambiguities. So far, one was accepted by the SCTP RFC committee.

Note that our model only captures two agents. Thus, it cannot be used to study DoS scenarios where a party might start an unbounded number of parallel SCTP connections. Our approach does not scale for such unbounded scenarios and a different approach would be needed. We discuss such approaches and how they compare to ours in Section 4.1.

Ethics. We disclosed all of our results to the chair of the SCTP RFC committee, resulting in an RFC erratum.

Code. All of our results are reproducible with our open source code, available at https://github.com/sctpfm.

2 SCTP

In this section we overview SCTP and previous efforts to validate it, as well as our approach to analyzing its security.

2.1 Overview

SCTP is a transport protocol proposed as an alternative to TCP, offering enhanced performance, security features, and greater flexibility. It is specified in several RFCs, each introducing significant modifications. RFC 9260 [58], which obsolesced RFC 4960 [55], made numerous small clarifications and improvements, including a critical patch for CVE-2021-3772. On the other hand, RFC 4960, which obsolesced the original specification in RFC 2960 [59], introduced major structural changes to the protocol as described in the errata RFC 4460 [15]. SCTP is implemented in Linux [2] and FreeBSD [4].

SCTP is a two peer protocol where each peer runs the same state machine. However, during connection establishment, the two peers play different roles – while one peer progresses through the *active* routine in the state machine, the other peer must take a corresponding sequence of *passive* transitions.[4] For teardown there are two options: graceful or graceless. During graceful tear-down, one peer can act actively and the other passively, or they can both take an active role. Graceless teardown happens in a single step.

Peer States. An SCTP peer is identified by a set of IP addresses and a port number. At any given time, each peer exists in one of finitely many states: Closed, in which no association exists; Cookie_Wait and Cookie_Echoed, used during active establishment; Established, in which an association exists and data can be transferred; Shutdown_Received and

[4] SCTP also supports an initialization routine where both peers are active, called "initialization collision". However, this routine is described in the RFC as an edge-case, rather than an intended use-case.

Shutdown_Ack_Sent, used by the passive peer during teardown; and Shutdown_Pending and Shutdown_Sent, used by the active peer during teardown. In active/active teardown, both peers use Shutdown_Ack_Sent.

Packets. An SCTP packet consists of a common header and a number of chunks. An essential component of the connection establishment design is authentication of packets between the peers using a random integer called the *verification tag*, or vtag, which is initialized using an *initiate tag*, or itag, during establishment. The packet header contains the source and destination port number, vtag, and a checksum. The chunk types are `INIT`, `INIT_ACK`, `COOKIE_ECHO`, and `COOKIE_ACK`, used during establishment; `DATA` and `DATA_ACK`, used for data transmission once an association has been established; `ERROR`, used to communicate when an error has occurred; `SHUTDOWN`, `SHUTDOWN_ACK`, and `SHUTDOWN_COMPLETE`, used during graceful teardown; `ABORT`, used during graceless teardown; and `HEARTBEAT` and `HEARTBEAT_ACK`, used for crash detection. Chunks contain parameters, e.g., `INIT` and `INIT_ACK` chunks (but no others) contain an itag, and (only) `INIT_ACK` chunks contain a *state cookie*, which includes a message authentication code, a timestamp indicating when the cookie was created, and a cookie lifespan. There are various kinds of `ERROR` chunks, each indicating a different error condition, e.g., `COOKIE_ERROR` which indicates receipt of a valid but expired state cookie. Much like TCP, SCTP uses sequence numbers, called Transmission Sequence Numbers (TSNs). The initial TSN in an association is proposed by an active participant in the connection establishment routine, and is incremented with each data transmission thereafter.

Connection Establishment. In active/passive establishment (Figure 1), the active peer sends a packet with an `INIT` chunk, containing a nonzero random itag. For the remainder of the association, this (active) peer will only accept packets from the passive peer that contain a vtag equal to the itag in the `INIT` it sent. The passive peer replies with a packet containing an `INIT_ACK` chunk, which also contains a nonzero random itag. For the remainder of the association, the passive peer will only accept packets which contain this itag value as the vtag in the common header. By checking the vtag, each peer protects itself from processing packets sent by an attacker not knowing the recipient's vtag.

Connection Teardown. Teardown can occur gracefully, via the active/passive or active/active routines, or gracelessly, with an `ABORT`. During active/passive teardown (Figure 2), the active peer sends a `SHUTDOWN` chunk, to which the passive peer responds with `SHUTDOWN_ACK`. The active peer then sends `SHUTDOWN_COMPLETE` and both transition to Closed. Active/active teardown is also possible, in which the peers exchange, in the following order: `SHUTDOWN`, `SHUTDOWN_ACK`, and `SHUTDOWN_COMPLETE` messages. The third option is that a peer can gracelessly abort a connection by sending an `ABORT` chunk. In this case, both peers immediately transition to Closed. Once the association is closed, the vtags are forgotten, and when either peer enters a new association, it will randomly choose a new itag (to become its vtag).

Timers. The SCTP connection routines use three timers: Init, Cookie, and Shutdown. The goal of the Init Timer is to stop the active peer in an establishment routine from getting stuck waiting forever for the passive peer to respond to its `INIT` with an `INIT_ACK`. The goal of the Cookie Timer is similar: it stops that same active peer from getting stuck waiting forever for the passive peer to respond to its `COOKIE_ECHO`. The Shutdown Timer plays a similar role but in the teardown routine, stopping the active peer in teardown from getting stuck waiting for a `SHUTDOWN_ACK`.

Out-of-the-Blue Packet Handling. In SCTP a message is considered *out-of-the-blue* (OOTB) if the recipient cannot determine to which association the message belongs, i.e., if it has an incorrect vtag, or is an `INIT` with a zero-valued itag. Specifically, an OOTB message will be discarded if: 1) it was not sent from a unicast IP, 2) it is an `ABORT` with an incorrect vtag, 3) it is an `INIT` with a zero itag or incorrect vtag[5], 4) it is a `COOKIE_ECHO`, `SHUTDOWN_COMPLETE`, or `COOKIE_ERROR`, and is either unexpected in the current state or has an incorrect vtag, or 5) it has a zero itag or incorrect vtag.

Unexpected Packet Handling. A message is *unexpected* if it is not OOTB, but nevertheless, the recipient does not expect it. SCTP handles unexpected packets as described in Algr. 1.

Algorithm 1 Unexpected Packet Handling

```
Require: Unexpected msg
  if msg.chunk = INIT then
    if state = Cookie_Wait or msg does not indicate new
    addresses added then
      Send INIT_ACK with vtag = msg.itag
    else
      Discard msg and send ABORT with vtag = msg.itag
    end if
  else if msg.chunk = COOKIE_ECHO then
    if msg.timestamp is expired then
      Send COOKIE_ERROR
    else if msg has fresh parameters then
      Form a new association
    else
      Set vtag = msg.vtag /* init collision */
      goto Established
    end if
  else if msg.chunk = SHUTDOWN_ACK then
    Send SHUTDOWN_COMPLETE with vtag = msg.vtag
  else
    Discard msg
  end if
```

Other Functionality. Other functionalities of SCTP include a "tie-tag" nonce mechanism used to authenticate a

[5]Per RFC 4960, respond with an `ABORT` having the vtag of the current association. But per RFC 9260, discard it.

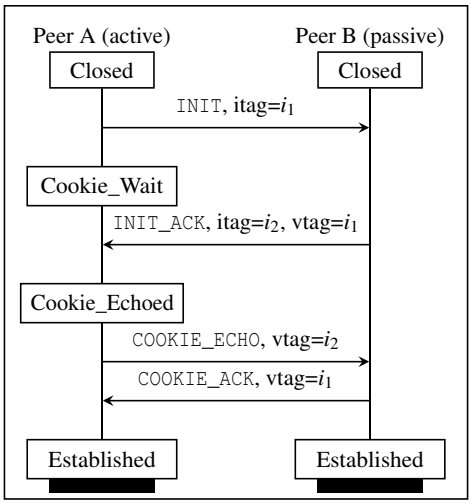

Figure 1: Message sequence chart illustrating SCTP active/-passive establishment routine. Time flows from the top down.

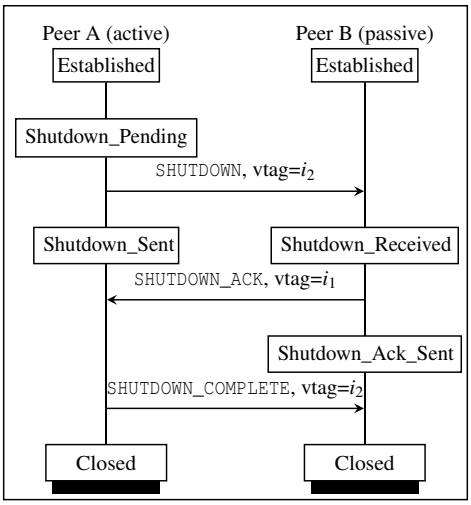

Figure 2: SCTP active/passive teardown.

reconnecting peer after a restart; congestion control[6]; fragmentation and reassembly of DATA chunks; chunk bundling; support for the Internet Control Message Protocol (ICMP); and multihoming. We do not consider this functionality in our analysis, and we refer the reader to [58] for more details.

2.2 Prior Validation

Conformance testing. The Linux and FreeBSD implementations were tested with PACKETDRILL [1] and fuzz-tests, suggesting they are crash-free and follow the RFCs. But this does not necessarily imply the *design* in the RFCs behaves correctly in the (a) absence or (b) presence of an attacker.

Formal analysis. For (a), some prior works formally analyzed SCTP using Colored Petri Net models [40, 62, 63, 68] in CPNTOOLS. This software can check for livelocks (i.e. liveness violations) and deadlocks (stuck states), but it cannot model-check arbitrary logical properties, which seriously limits the use-cases for such models. One prior work studied (b), modeling the four-way handshake used by SCTP and comparing it to the three-way handshake used by TCP in the presence of an attacker, with the Uppaal model-checker [53]. However, the model is closed-source and does not include the teardown routine. It is unclear whether the model includes OOTB or unexpected packet handling. We summarize the differences between these prior models and our own in Table 1. Finally, the IETF published a security memo for SCTP, but it is not a comprehensive analysis, rather, it simply summarizes prior conversations about security from the SCTP user-group [56].

CVE-2021-3772 attack and patch. As reported in CVE-2021-3772 [49], the prior version of SCTP specified in RFCs 2960 [59] and 4960 [55] is vulnerable to a denial-of-service attack. The reported vulnerability worked as follows. Suppose SCTP peers A and B have established a connection and an off-channel attacker knows the IP addresses and ports of the two peers, but not the vtags of their existing connection. The attacker spoofs B and sends a packet containing an INIT to A. The attacker uses a zero vtag as required for packets containing an INIT. The attacker must use an illegal parameter in the INIT, e.g., a zero itag.

Peer A, having already established a connection, treats the packet as out-of-the-blue, per RFC 2960 §8.4 and 5.1, which specify that as an association was established, A should respond to the INIT containing illegal parameters with an ABORT and go to Closed. But in RFCs 2960 and 4960, it is unspecified which vtag should be used in the ABORT. Some implementations used the expected vtag, which is where a vulnerability arises. Since the attacker spoofed the IP and port of Peer B, Peer A sends the ABORT to Peer B, not the attacker. When Peer B receives the ABORT, it sees the correct vtag, and tears down the connection. Thus, by injecting a single packet with zero-valued tags, the attacker tears down the connection, pulling off a DoS. The attack is illustrated in Figure 3.

RFC 9260 patches CVE-2021-3772 using a strict defensive measure, wherein OOTB INIT packets with empty or zero itags are discarded, without response. FreeBSD [4] uses this patch. Linux, on the other hand, adopts a different patch [39], wherein the peer receiving the ABORT with the zero vtag simply ignores it (rather than close the connection).

3 Our SCTP Model

In this section, we describe our SCTP PROMELA model and properties that guide our analysis.

[6](based on TCP congestion control)

Work	RFC	Open-Source	Establish	Teardown	OOTB	Unexpected	Livelocks	Deadlocks	Properties
Martins et. al. [40]	2960	N	Y	Y	N	N	Y	Y	N
Blanchet et. al. [68]	2960	N	Y	Y	N	N	Y	Y	N
Vanit-Anunchai [62]	4960	N	Y	Y	Y	Y	Y	Y	N
Vanit-Anunchai [63]	4960	N	Y	Y	Y	Y	Y	Y	N
Saini and Fehnker [53]	4960	N	Y	N	N	N	Y	Y	Y
Ours	4960 & 9260	Y	Y	Y	Y	Y	Y	Y	Y

Table 1: Prior formal SCTP analyses versus ours. RFC column reports modeled version, and open-source column reports whether the model is open-source. The remaining columns report whether the model includes the establish and teardown routines, OOTB logic, or unexpected packet handling; and if it can be used to check for livelocks or deadlocks, or to verify arbitrary properties.

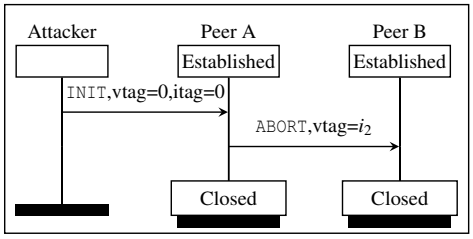

Figure 3: CVE-2021-3772 Attack. Peers A and B begin having established an association with vtags i_1, i_2 (resp.). The Attacker transmits an invalid `INIT` chunk to A, spoofing the port and IP of B. Peer A responds by sending a valid `ABORT` to B, which closes the association. By sending a single invalid `INIT` the Attacker performs a DoS.

3.1 Overview

As we are primarily interested in denial-of-service attacks, and in order to avoid state-space explosion, we selectively model the SCTP connection establishment and teardown routines. This allows us to automatically and exhaustively explore, simulate, and verify the critical, security-relevant aspects of SCTP. Our model captures the following aspects of SCTP per RFC 9260 [58]: internal peer states, packet verification using the itag and vtag, timers, TSNs, and handling for invalid, unexpected, and OOTB packets. We made only the abstractions listed in Section 3.4.

Although our model is fully faithful to the SCTP RFC [58], and is an executable program, it is not a network library and cannot be used in place of the existing Linux or FreeBSD implementations. This is because of the abstractions and simplifications mentioned above, and also because it does not implement API hooks for higher-level applications, nor syscalls to transmit over the Internet. It is simply a model of SCTP with which we can formally verify correctness properties.

3.2 Model Details

We describe our SCTP model in PROMELA, focusing on internal peer states, packet verification, invalid packet defense mechanisms, timeouts, and OOTB packet handling.

Mathematical Preliminaries. Linear Temporal Logic is a modal logic for reasoning about program executions. In LTL, we say a program P *models* a property ϕ, written $P \models \phi$, if ϕ holds over every execution of P. If ϕ holds over some but not all executions of P, then we write $P \not\models \phi$.

The LTL language is given by predicates (e.g., "Peer A is in Established" or "Peer B's cookie timer is inactive"); the temporal operators "next", "always", "eventually", and "until"; and the logical operators of negation, conjunction, and disjunction. An LTL model-checker is a tool that, given P and ϕ, can automatically check whether or not $P \models \phi$ [7]. We use the model-checker SPIN[8], whose language is PROMELA.

We use $\|$ to denote rendezvous composition, so, $S = P \| Q$ denotes that the program S equals the composition of P with Q. Specifically, matching *send* transitions of P and *receive* transitions of Q occur in lockstep, and vice versa. Note, in our model, we actually build a process called a "channel" to capture network delay, and we rendezvous-compose the channel with the two peers to build asynchronous communication (which is more realistic). The $\|$ operator is commutative and associative. For more details, refer to §2 of [67].

Internal Peer States. Our model consists of two peers (A and B) and a channel connecting them. That is, we study the system $S = \text{PEERA} \| \text{CHANNEL} \| \text{PEERB}$. Each peer is represented by an identical FSM, illustrated in Figure 5. Transitions between states occur based on the receipt of user commands, or communication and message processing.

The channel connecting the two peers contains an internal single-message buffer in each direction (meaning it can hold two messages at once, one traveling from left to right and the other from right to left). It does not drop, corrupt, nor create messages, and cannot accept a new message in a given direction until the old one was delivered. In other words, it is lossless and FIFO, in that it guarantees every delivered message was sent and messages are delivered in order. The entire setup is illustrated in Figure 4.

Packet Verification and Invalid Packet Defenses. We model each SCTP message as consisting of a message chunk, a vtag, and an itag. Each of these components are modeled using enums, which in PROMELA are called `mtypes`. The message

[7] LTL model-checking is decidable, and reduces to checking Büchi Automata intersection emptiness, which is PSPACE-complete.

[8] version 6.5.2

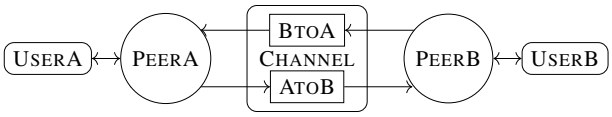

Figure 4: The system UserA || PeerA || Channel || PeerB || UserB. Channel contains a size-1 FIFO buffer in each direction (AtoB and BtoA, respectively). Arrows indicate communication direction. Composition between Channel and peers is rendezvous (the buffers are inside Channel).

chunk denotes the meaning of the message, e.g., a message with an `INIT` chunk is called an *initiate message* and is used to initiate a connection establishment routine. The itag and vtag are used to verify the authenticity of the sender of the message, as described in Section 2.2. In our model there are three kinds of tags: expected (`E`), unexpected (`U`), or none (`N`). A tag is expected if (1) it is a non-zero itag on an `INIT` or `INIT_ACK` chunk, or (2) it is the other peer's vtag in the existing association. Otherwise, it is unexpected. The none type is reserved for packets that do not carry the given tag type – e.g., only `INIT` and `INIT_ACK` chunks carry an itag, so in the other types of messages, the itag is `N`. The BNF grammar for messages in our model is given in Figure 6. We also support an option where the *msg* can be extended with a TSN.

Upon receiving a message, our model checks that the tags are set as expected, depending on the message and state. If a message has an unexpected tag then the model employs the defenses specified in the RFC, e.g., silently discarding the message or responding with an `ABORT`. These defenses can be configured with or without the CVE patch from RFC 9260.

Active and Passive Connection Routines. Our SCTP model implements active/passive establishment and teardown, as well as active/active teardown, but not active/active establishment (a.k.a. "`INIT` collision"), precisely as described in Section 2 and illustrated in Figures 1 and 2, with the caveat that the itag and vtag are abstracted (as described above). We also capture the TSN proposal and use throughout an association, although this feature can be turned off in our model to reduce the state-space for more efficient verification.

Out-of-the-Blue and Unexpected Packets. Our model faithfully captures OOTB logic described in §8.4 of RFCs 4960 and 9260, with only the exceptions given in Section 3.4.

3.3 Ambiguities in the RFC

While reading 9260 [58] in order to build our model, we found two ambiguities. In each case, we reported the ambiguity to the lead SCTP RFC author, confirmed the correct interpretation of the ambiguous text with them, and suggested a clarifying erratum.

The first ambiguity is in §5.2.1, during the description of how a peer should react upon receiving an unexpected `INIT`:

> *Upon receipt of an `INIT` chunk in the Cookie_Echoed state, an endpoint MUST respond with an `INIT_ACK` chunk using the same parameters it sent in its original `INIT` chunk (including its Initiate Tag, unchanged), provided that no new address has been added to the forming association.*

Consider two peers - A and B - initially both in Closed, in addition to some attacker who can spoof the port and IP of B. Suppose these machines engage in the sequence of events illustrated in Figure 7. At the end of the sequence, what value should the vtag V take?

A problem arises if a reader interprets "the same parameters" to include the vtag, implying that the vtag of the `INIT_ACK` should come from the `INIT` that the responding endpoint sent. Then V should take the itag of the message, i.e. $V = i_1$. The fact that this is wrong (and the correct assignment is $V = i_2$) only becomes clear if you fully understand how itags and vtags are used in both directions. To make the text unambiguous, we suggest the following erratum:

> The verification tag used in the packet containing the `INIT_ACK` chunk MUST be the initiate tag of the newly received `INIT` chunk.

The second ambiguity we identify is in §8.5, which says:

> *When receiving an SCTP packet, the endpoint MUST ensure that the value in the Verification Tag field of the received SCTP packet matches its own tag.*

The problem is that §8.5 does not say *when* the vtag check should happen with respect to other checks. In particular, §3.3.3 says that an endpoint in Cookie_Wait who receives an `INIT_ACK` with an invalid itag should respond with an `ABORT`– but it is unclear whether this still applies before or after the vtag check in §8.5. Under the former interpretation, an endpoint in Cookie_Wait who receives an `INIT_ACK` with both an invalid itag and an invalid vtag would respond with an `ABORT`, whereas under the latter interpretation, the endpoint would silently discard the packet. To clarify the ambiguity, we proposed the following erratum, which the SCTP RFC committee accepted [61]:

> When receiving an SCTP packet, the endpoint MUST first ensure that the value in the Verification Tag field of the received SCTP packet matches its own tag before processing any chunks or changing its state.

3.4 Abstractions and Limitations

Our model is fully faithful to the SCTP RFC, modulo the following abstractions and limitations.

- **Unicast peers.** In the RFC, OOTB messages from non-unicast peers are discarded; we model all peers as unicast.
- **No crashes or restarts.** In our model, peers never crash or restart. Thus we also omit crash detection (including `HEARTBEAT` and `HEARTBEAT_ACK` chunks).
- **Tags are abstracted.** We do not model tie-tags, which are used when reconnecting a peer to an existing association after a restart. In the RFC, itags and vtags are integer-valued and chosen randomly. But we model tags as the "expected" value,

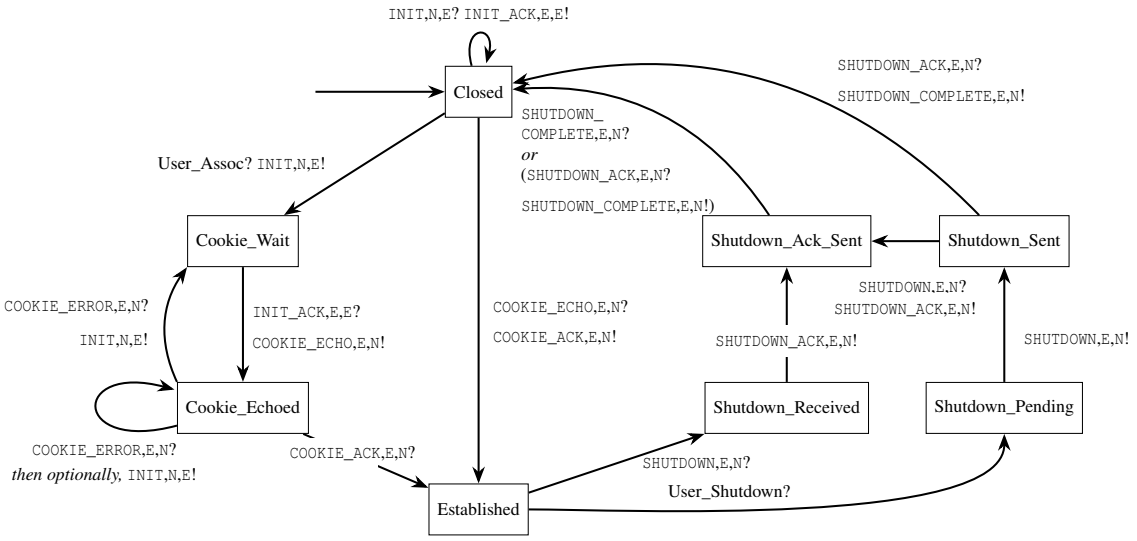

Figure 5: SCTP Finite State Machine. $x,v,i?$ (or $x,v,i!$) denotes receive (or send) chunk x with vtag v and itag i. Events in multi-event transitions occur in the order they are listed. Logic for OOTB packets, ABORT messages or User_Abort commands, unexpected user commands, and data exchange are ommitted but faithfully implemented in the model and described in this paper.

$$
\begin{aligned}
msg &::= \texttt{INIT},\texttt{N},ex \mid \texttt{INIT\_ACK},ex,ex \mid ach,ex,\texttt{N} \\
ach &::= \texttt{ABORT} \mid \texttt{SHUTDOWN} \mid \texttt{SHUTDOWN\_COMPLETE} \\
&\quad \mid \texttt{COOKIE\_ECHO} \mid \texttt{COOKIE\_ACK} \mid \texttt{SHUTDOWN\_ACK} \\
&\quad \mid \texttt{COOKIE\_ERROR} \mid \texttt{DATA} \mid \texttt{DATA\_ACK} \\
ex &::= \texttt{E} \mid \texttt{U}
\end{aligned}
$$

Figure 6: BNF grammar for messages in our model.

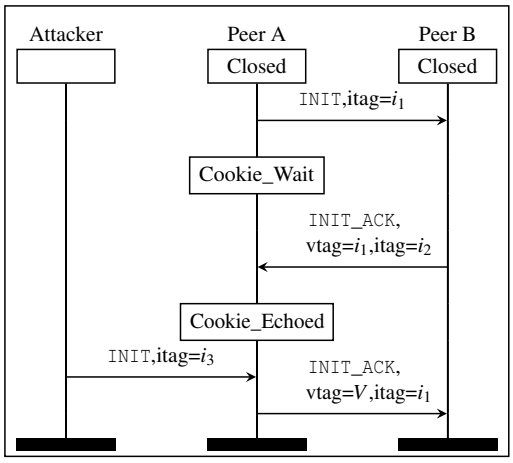

Figure 7: First ambiguity. What value should V take? See Section 3.3.

an "unexpected" value, or "none", since this level of detail is all that matters for our properties. A side-effect is that we cannot study INIT collision. INIT collision is not included in the State Association Diagram in RFC 9260 §4, nor in the various association flows throughout the RFC document, leading us to believe it is not a protocol feature but rather an edge-case the protocol is designed to withstand.

• **Perfect channel.** We do not model packet loss, reordering, nor corruption, nor how SCTP deals with these scenarios.

• **Peers do not exchange data while in Established.** Because we focus on denial-of-service attacks, modeling data exchange while in Established is unnecessary; rather, we focused on the connection and disconnection of peers. We did model data transmission outside of Established, in case it caused edge-case behaviors during teardown.

• **Packets only ever contain one chunk.** Since we also do not model (or write properties about) fragmentation, bundling, or reassembly, we can simulate multi-chunk transmissions by sending consecutive single-chunk messages.

• **Simplified packet structure.** We choose not to model packet structure details relevant to only DATA packets, e.g.: stream sequence number, payload protocol identifier, and variable length. We also do not model ICMP messages.

3.5 Correctness Properties

Next we transcribe ten logical properties we believe SCTP should satisfy. Note, we do not intend to create a *complete* set of properties that captures all behaviors of SCTP. Rather, we design our properties to capture the security-relevant behavior of SCTP. Each property is implemented in PROMELA using

LTL. We justify each using the RFCs [55,58] and our intuition about the security SCTP should provide.

ϕ_1**: A peer in Closed either stays still or transitions to Established or Cookie_Wait.** This is based on the routine described in §5.1, as well as the Association State Diagram in §4. If a closed peer could transition to any state other than Established or Cookie_Wait, it could de-synchronize with the other peer, breaking the four-way handshake and potentially leading to a deadlock, livelock, or other problem.

ϕ_2**: One of the following always eventually happens: the peers are both in Closed, the peers are both in Established, or one of the peers changes state.** The property we want to capture here, "no half-open connections", is stated in §1.5.1, was verified in the related work by Saini and Fehnker [53], and was studied for TCP in two prior works [47, 67]. But we have to formalize it subtly, because in the case of an in-transit `ABORT`, it is possible for one peer to temporarily be in Established while the other is in Closed; so we write it as a liveness property, saying half-open states eventually end.

ϕ_3**: If a peer transitions out of Shutdown_Ack_Sent then it must transition into Closed.** We derived this from the Association State Diagram in §4. Every transition out of Shutdown_Ack_Sent described in the RFC ends up in either Closed or Shutdown_Ack_Sent. If this property fails, it would imply a flaw in the graceful teardown routine, and could cause a deadlock, livelock, or other problem.

ϕ_4**: If a peer is in Cookie_Echoed then its cookie timer is actively ticking.** Per §5.1 C), the peer starts the cookie timer upon entering Cookie_Echoed. Per §4 step 3), when the timer expires it is reset, up to a fixed number of times, at which point the peer returns to Closed. If the property fails, then the active peer in an establishment could get stuck in Cookie_Echoed forever, opening a new opportunity for DoS.

ϕ_5**: The peers are never both in Shutdown_Received.** This property follows from inspection of the Association State Diagram in §4. From a security perspective, if both peers were in Shutdown_Received, this would indicate that neither initiated the shutdown (yet both are shutting down); the only logical explanation for which is some kind of DoS.

ϕ_6**: If a peer transitions out of Shutdown_Received then it must transition into either Shutdown_Ack_Sent or Closed.** The transition to Shutdown_Ack_Sent is shown in the Association State Diagram in §4. The transition to Closed can occur upon receiving either a User_Abort from the user or an `ABORT` from the other peer. No other transitions out of Shutdown_Received are given in the RFC. If this property fails, it could de-synchronize the teardown handshake, potentially leading to an unsafe behavior. For example, if a peer transitioned from Shutdown_Received to Established, it would end up in a half-open connection.

ϕ_7**: If Peer A is in Cookie_Echoed then B must not be in Shutdown_Received.** We derived this from the Association Diagram in §4, which shows A must receive an `INIT_ACK` while in the Cookie_Wait and then send a `COOKIE_ECHO` in order to transition into Cookie_Echoed. B must have been in Closed to send an `INIT_ACK` in the first place, hence B cannot be in Shutdown_Received. This property relates to the synchronization between the peers: if one is establishing a connection while the other is tearing down, then they are de-synchronized, and the protocol has failed.

ϕ_8**: Suppose that in the last time-step, Peer A was in Closed and Peer B was in Established. Suppose neither user issued a User_Abort, and neither peer had a timer time out. Then if Peer A changed state, it must have changed to either Established, or the implicit, intermediary state in Cookie_Wait in which it received `INIT_ACK` but did not yet transmit `COOKIE_ECHO`.** The transitions from Closed to Established and the described intermediary state are implicit in the Association State Diagram in §4. The timer caveat is described in §4 step 2, and the aborting caveat is in §9.1. If the property fails, the four-way handshake ended, yet was not completed successfully, did not time out, and was not aborted, so somehow, the protocol failed.

ϕ_9**: The same as ϕ_8 but the roles are reversed.** The property is: *Suppose that in the last time-step, Peer B was in Closed and Peer A was in Established...*

ϕ_{10}**: Once connection termination initiates, both peers eventually reach Closed**. This follows from the description of connection termination in §9. Once connection termination is initiated, there is no way to recover the association.

For the On-Path attacker model, ϕ_8 and ϕ_9 are symmetric. For the other attacker models, the properties are distinct, because the attacker model's network topology is asymmetric.

3.6 Validating Our Model

Our model allows us to execute and reason about any component of the peer logic in isolation, or two interacting peers. To verify our model, we extracted the properties listed above from the SCTP RFCs, and then used the model-checker to prove that our model satisfies all of the properties. We interactively guided SPIN to drive the model through various connection flows (which we compared to the RFC text), and we manually compared our logic for handling OOTB packets to the corresponding C code in Linux and FreeBSD. Finally, we used SPIN to prove there were no deadlocks or livelocks (liveness cycles) and all the peer states are reachable.

4 SCTP Attack Synthesis

In this section we provide details on attack synthesis and KORG, the tool we used. Next, we describe four attacker models we defined and used for our analysis. Finally, we present the changes we had to make to KORG to handle our SCTP model, and the four attacker models we considered.

4.1 Attack Synthesis

LTL program synthesis is the problem of, given an LTL specification ϕ, automatically deriving a compliant program P (for which $P \models \phi$). *LTL attack synthesis* is fundamentally different (logically dual), and cannot be solved using program synthesis alone. In attack synthesis, the problem is flipped: given a program S and property ϕ, where S is already compliant ($S \models \phi$), if $S = P \parallel Q$ consists of an *invariant component* P (that the attacker cannot change) and a *variant component* Q (that the attacker can change), we ask whether there exists some modification A such that, if we replace Q with A, the new system $S' = P \parallel A$ is non-compliant ($S' \not\models \phi$). In other words, we study a system that behaves correctly, and ask if we can change some constrained aspect so that it behaves incorrectly. If so, we call this modification A an "attack".

There are multiple kinds of attacks one might try to synthesize, depending on the nature of the protocol and the attacker goal. We use use KORG [67], which leverages SPIN [28] to synthesize attacks against arbitrary LTL properties of transport protocols. LTL includes safety properties – which say that something bad never happens (e.g., ϕ_4) – and liveness properties – which say that something good eventually happens (e.g., ϕ_{10}). The main difference between SPIN and verifiers such as PROVERIF [13] and TAMARIN [44] is that SPIN only supports finite-state models and LTL, whereas PROVERIF and TAMARIN support infinite-state models, and other kinds of properties such as forward secrecy and trace equivalence. However, this expressiveness comes at a cost, which is that the unbounded verification of liveness properties in the cryptographic verification setting is undecidable [7].[9] Cryptographic verifiers can verify liveness properties in a bounded fashion, i.e., by confirming that something good happens within a bounded number of steps (TAMARIN does this [8]); or in an unbounded fashion, by relying on human input in the form of proof hints (see e.g., [60, 70]).

In contrast, an LTL model checker such as SPIN can only reason about finite state-spaces, but, can automatically verify both safety and liveness properties of finite systems (such as the SCTP handshake) in an unbounded fashion, by exploiting the omega-regularity of LTL over finite Kripke structures [28]. KORG is proven to be sound (it has no false-positives) and complete (if attacks of the kind KORG looks for exist, given enough resources, KORG will find one) [67]; with the caveat that it is restricted to small, finite-state systems with LTL specifications. Since the SCTP handshake does not use any cryptographic primitives other than the cookie, and can be described using a small, finite-state model, and since we want to determine whether any attacks whatsoever exist against certain properties (and not just attacks of a certain size), we choose KORG for our analysis; but if we aimed to study a cryptographic or infinite-state protocol such as TLS or Signal (see: [17, 36], resp.), we would need a verifier such as PROVERIF or TAMARIN, or a hybrid approach combining multiple tools, as in [18]. Note, KORG was previously successfully applied to TCP and DCCP [47, 67], and adds the attack synthesis functionality to SPIN, which itself has existed for 35 years; has been applied to dozens of real systems including the Mars rover [29], PathStar access server [30], and ISO/IEEE P11073-20601 medical communication protocol [24]; spawned a dedicated formal methods conference, currently in its 30th year[10]; and earned the 2002 ACM Software System Award.

KORG requires four inputs: an invariant component P (e.g., the SCTP model) and variant component Q (which in our case is part of the attacker model), both in PROMELA; an LTL correctness property ϕ, such that the composite system consisting of both P and Q satisfies ϕ ($P \parallel Q \models \phi$); and a YAML file encoding the grammar (I/O) of Q (which become the I/O of the attacker). KORG generates an model with these inputs in which Q is replaced with a process called a *daisy*, that can nondeterministically send or receive messages specified in the grammar. Next, it modifies ϕ to have a precondition saying the daisy terminates, and then asks SPIN to verify or disprove the modified property for the modified model. Either KORG reports no attacks exist, or SPIN outputs a counterexample execution, which KORG parses into an attack A. For more refer to [67]. The inputs to KORG needed to reproduce each of our experiments are documented in the paper artifacts.

KORG is limited by the level of detail in the model, the definition of "attack" used by KORG [67], and the attacker models and properties considered. Thus there could exist other attacks beyond those KORG synthesizes, which violate other properties or work in other attacker models; or attacks other than the type that KORG can find (e.g., statistical ones); or attacks that cannot be found without a more detailed model. These limitations are inherent to all attack synthesis techniques.

4.2 Attacker Models

We define an *attacker model* to be a formal description of the placement and capabilities of the attacker and protocol peer(s) on the network. We create four attacker models: Off-Path, Evil-Server, Replay, and On-Path. They are general-purpose and applicable to any transport protocol, and we contribute them to KORG. Of these, SCTP was only designed to be fully robust against an Off-Path attacker; we study the others to understand what could happen in a "worst-case" scenario. Note, we model only DoS attacks with two agents. Our approach does not scale for scenarios with unbounded parallel sessions, such as Dolev-Yao.

Off-Path. This is the primary attacker model discussed in the IETF SCTP security memo [56] and is what SCTP was designed to be fully secure against. In this model, an attacker

[9] Indeed, even simple reachability problems such as secrecy are not in general decidable for finite, straight-line, role descriptions in PROVERIF or TAMARIN, and so often the human using the tool must supply proof hints.

[10] https://spin-web.github.io/SPIN2023/

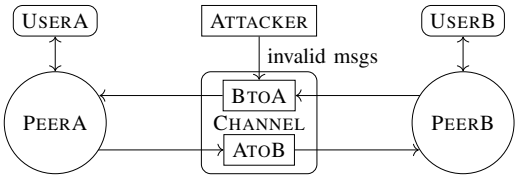

Figure 8: Off-Path Attacker Model: S = ATTACKER ‖ USERA ‖ PEERA ‖ CHANNEL ‖ PEERB ‖ USERB. The attacker can transmit messages into the BtoA buffer, but cannot receive messages, nor block messages in-transit. The attacker can send only chunks having an invalid itag and vtag (as it is not privy to the association).

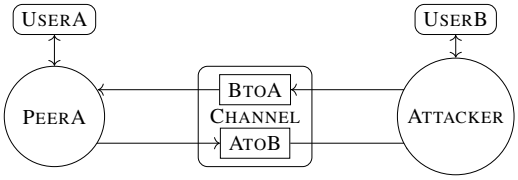

Figure 9: Evil-Server Attacker Model: S = USERA ‖ PEERA ‖ CHANNEL ‖ ATTACKER ‖ USERB. Peer B is prefixed with an attacker, whose code consists of a finite, terminating sequence of communication operations.

who does not know either vtag communicates with one peer in order to disrupt a connection between the two peers. The vtag mechanism in SCTP was designed to defend against such an attacker. See Figure 8.

Evil-Server. In this model, one of the peers behaves maliciously. The attacker takes the form of a finite sequence of malicious instructions inserted before the code of Peer B, after which B behaves like normal. The purpose of this attacker model is to study the degree to which an adversary capable of temporarily controlling one peer in a connection can negatively impact the other. For example, could it cause the second peer to deadlock? See Figure 9.

Replay. In this model (Figure 10), the attacker can replay captured packets without modification. SCTP was designed to withstand most replay attacks using its TSN mechanism, as discussed in IETF TSVWG 115 [42].

On-Path. In this attacker model, the attacker controls the

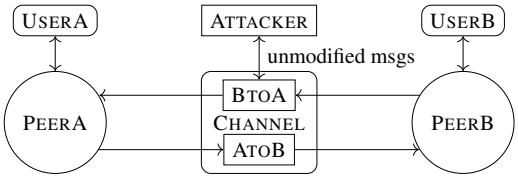

Figure 10: Replay Attacker Model: S = ATTACKER ‖ USERA ‖ PEERA ‖ CHANNEL ‖ PEERB ‖ USERB. The attacker can capture and re-transmit messages in BtoA, but cannot edit captured messages, nor block those in transit.

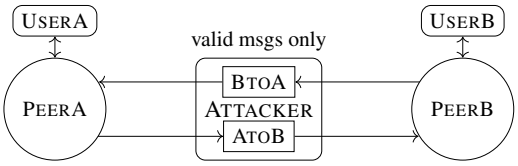

Figure 11: On-Path Attacker Model: S = USERA ‖ PEERA ‖ ATTACKER ‖ PEERB ‖ USERB. The attacker is allowed to perform a finite sequence of send/receive actions, in which it only sends valid messages (but can receive anything). Once this sequence terminates, it behaves like an honest channel.

channel connecting the two peers, and can drop or insert valid messages at-will. SCTP was not designed to provide security against such an attacker and we study this attacker model only to understand what the "worst case scenario" for SCTP looks like (e.g., as studied in [34]). See Figure 11.

4.3 Changes to KORG

We improved KORG to support our SCTP analysis in four ways. (1) Since KORG was originally hard-coded for enum-style packets, we extended KORG to support arbitrary finite packet types. This was needed to support our SCTP model (Figure 6). (2) We modified KORG to report any attacks it finds even if it fails to exhaust the search-space. Previously, it would report an error and discard any results if the space was not exhausted. (3) To save time, we disabled the preliminary step where KORG verifies that the property holds in the absence of an attacker, instead manually performing this step in SPIN. (4) We extended KORG to support replay attackers.

A replay attacker is one capable of capturing and replaying messages. Although the replay attacker model reasons about packets received, the attacks this model produces form a subset of those produced by the *On-Path* attacker; thus, soundness and completeness follow from the proofs in the KORG paper. Our replay attacker synthesis implementation supports packet capture and replay over the same or different channels. In the latter case, the attacker can capture a message from one channel and replay it into another. It also supports packet storage in a memory buffer with configurable size, though, the verification complexity increases exponentially with the memory bound. Finally, to support the state change that happens when a new vtag is chosen, we added a feature where a special message can be configured to flush the storage.

We also contribute our SCTP model and four attacker models in a format amenable to KORG. We document the attacker models in Section 4.2. Excluding the models, our modifications required changing 80 lines of preexisting code and adding 213 lines of new functionality in KORG, in addition to 102 lines of shell-script to automate our experiments. All modifications are available with the paper artifacts.

5 Experimental Results

We next present our experimental results. Our SCTP model satisfies all ten properties in the absence of an attacker. To examine whether these properties still hold when an attacker is present, we synthesize attacks using the four attacker models. Then, we enable the CVE patch described in RFC 9260 and repeat our analysis, in order to check whether the patch resolves the vulnerability, and/or introduces any new attack vectors. Finally, we show how new attacks are enabled if either ambiguity in Section 3.3 is misinterpreted. Analysis runtimes are related in the appendix.

5.1 Experimental Methodology

Each time we run KORG, we ask it to synthesize ≤ 10 attacks. In our experience, after the first ten, subsequent attacks tend to be repetitive, differing only by actions that do not impact the attack outcome. We configure KORG with a default search depth of 600,000, and a maximum depth of 2,400,000. In our experience, these parameters balance fast-performance on smaller properties with the ability to also attack more complex ones, without needing to run on a cluster. While KORG may run out of time or memory without finding an attack for larger finite state machines, in the experiments presented in this section, KORG never runs out of memory. It either finds at least one attack or fully exhausts the search-space before hitting the configured maximum search depth. Recall, however, that we only study attacker models involving two agents (i.e., one association). KORG does not scale to larger attacker models such as Dolev-Yao, which might involve an unbounded number of agents. We make certain assumptions or optimizations in the different attacker models.

• *Off-Path:* We assume the Off-Path attacker knows the port and IP of a peer, since otherwise, all its (spoofed) messages will be immediately discarded.[11] To reduce the search-space, we assume the attacker does not send `DATA` or `COOKIE_ERROR` chunks, which cannot change the receiving peer's state. We further reduce the space by first synthesizing attacks against the establishment routine, where the attacker could only send messages that are used during establishment; and then doing the same for teardown. Our search is complete despite this split because the FSM is inherently Markovian and our properties do not look back more than one state in the past. In our open-source artifacts, we provide code illustrating how this optimization can be repeated for any transport protocol.

• *Evil-Server:* We assume the Evil-Server attacker only sends valid messages, since it knows the current vtags. To reduce the search-space, we assume it does not send `DATA`.

• *Replay:* We configure the replay attacker to have a memory size of two — we find that more memory causes state-space explosion making exhaustive verification infeasible. We also configure it to discard all messages in memory upon receiving an `INIT`, allowing us to correctly model the vtag change between multiple connection and teardown cycles.

• *On-Path:* We perform the same optimizations as in the Off-Path attacker model. And like in the Evil-Server attacker model, we assume the attacker only sends valid messages.

[11] The ports and IP of a peer might not change between associations [57].

```
BTOA ! COOKIE_ACK, U, N;        (repeat twice more)
BTOA ! COOKIE_ECHO, U, N;
BTOA ! COOKIE_ACK, U, N;        (repeat 6 more times)
BTOA ! COOKIE_ECHO, U, N;
BTOA ! INIT, N, U;                    /* attack */
```

Figure 12: Automatically synthesized CVE attack in the Off-Path attacker model. BTOA is the channel from the attacker to the peer being attacked. Only the final line matters.

5.2 Attacks

We generate at least one attack in each attacker model, all of which we summarize in Table 2. However, recall that SCTP was only designed to withstand Off-Path attacks, so, results in the other attacker models do not negatively impact our perspective on the security of SCTP. And in the Off-Path model, we find only the CVE attack with the patch is disabled, and no attacks when the patch is enabled. Therefore, our results can be seen as affirming the security of SCTP. We discuss results for each attacker model in detail below.

Off-Path. KORG found a variant of the attack reported in CVE-2021-3772, given in Figure 12. The variant differs only from the CVE in that it begins by transmitting some OOTB messages that are discarded and have no impact on the outcome. It ends with the transmission of an `INIT` with an unexpected (zero) itag, which is the CVE attack.

Evil-Server. KORG synthesizes four attacks. The first attack models a scenario in which the Shutdown Timer is configured to a very large value, and thus the attacker can cause a peer in active teardown to (essentially) deadlock by never responding to its `SHUTDOWN` message. In the second attack, the attacker exploits the unexpected packet logic in §5.2.4 to guide a peer out of passive teardown and back into Established. The third and fourth attacks are similar and involve guiding one peer through establishment to de-synchronize it with the other, leading to a half-open connection. Although SCTP was not designed to withstand an Evil-Server, it can nevertheless be configured to defend against attacks like those we found through the appropriate configuration of timers.

Replay. KORG synthesizes one attack where the attacker captures and replays an `ABORT` message sent by a peer before both peers establish a new TSN. The attacker can keep replaying the `ABORT` message indefinitely, preventing the peers

Model	Prop.	Synthesized Attacks	Category
Off-Path	ϕ_9	1 variant of the CVE attack.	Attack
	ϕ_8	1 opportunistic `INIT` attack (Fig. 14).	Misinterpretation
	ϕ_4	1 opportunistic `INIT_ACK` attack (Fig 15).	Misinterpretation
Evil-Server	ϕ_1	1 where the attacker guides A through passive establishment. Then when A attempts active teardown, if its Shutdown Timer never fires, it deadlocks.	Misuse
	ϕ_6	1 where the attacker guides A to Shutdown_Received, then sends it an unexpected `COOKIE_ECHO`, causing it to go back to Established.	Misuse
	ϕ_8	1 where the attacker guides A through most of active establishment before aborting the connection. When the attack terminates, B receives the en-route `COOKIE_ECHO` and completes passive establishment, creating a half-open connection.	Misuse
	ϕ_9	1 where the attacker guides A through passive establishment then terminates. If B then attempts active establishment, the property fails, since the peers are de-synchronized.	Misuse
Replay	ϕ_2	1 where the attacker sends an `ABORT` before the peers establish a TSN for the association.	Misuse
On-Path	ϕ_5	4 attacks where the attacker manipulates both peers into Shutdown_Received.	Misuse
	ϕ_8	2 where the attacker spoofs A to guide B through passive establishment.	Misuse
	ϕ_9	2 where the attacker spoofs B to guide A through passive establishment.	Misuse

Table 2: Attacks found without the patch.

from establishing a connection. This attack is hard to pull off because it requires one of the peers to first `ABORT` and then attempt to re-establish a connection. No other attacks were found. This is expected, as the OOTB logic and TSNs should prevent a replay attacker from injecting old packets.

On-Path. KORG synthesizes four similar attacks where the attacker guides both peers into an association, and then spoofs each peer, sending a `SHUTDOWN` to the other. In general, an On-Path attacker is so powerful that we expect it can manipulate either peer into any state it pleases, as it totally controls the network, so this is unsurprising. KORG synthesizes two more attacks, one for each of the half-open properties, both similar to the last attack reported with the Evil-Server attacker model. These results reinforce the IETF SCTP security memo's statement that "only a strong end-to-end security model can prevent" On-Path attacks.

Note, the reason we do not rediscover the CVE attack in the Evil-Server or On-Path attacker model is that we restrict the attacker in both to only send valid messages, whereas the CVE attack requires an invalid `INIT`. We put this restriction in place in our model to avoid state-space explosion. In general, every attack that is possible in the Off-Path attacker model is also possible in the Evil-Server and On-Path ones.[12]

5.3 Patch Verification

Next, we re-run our analysis with the CVE patch enabled. In the Off-Path attacker model, KORG terminates without finding any attacks. Since KORG found the vulnerability in the Off-Path attacker model when the patch was disabled, and reports no attacks in that same attacker model when the patch is enabled, and as KORG is complete, this suffices to prove that the patch resolves the vulnerability. In the other attacker models, we find the exact same attacks as those reported in Table 2, and nothing more. This proves the patch does not decrease the security of the protocol as it pertains to attacks in any of the four attacker models we study and against the ten LTL properties we specify. However, it does not mean that no new attacks are introduced outside the scope of our analysis, such as, statistical or side-channel attacks or attacks against other properties or in other attacker models besides those listed. Also, our analysis is inherently limited to the attacker models we study, in which the attacker can only perform DoS attacks against a single association involving two agents.

[12](up to isomorphism)

5.4 Ambiguity Analysis

Next, for each of the two ambiguities, we configure our model with the incorrect interpretation thereof and synthesize attacks against it. We find that the incorrect interpretation of either ambiguous portion could enable a DoS attack, which we illustrate in Figure 14 and Figure 15, respectively, in the Appendix. We consulted with the lead SCTP RFC author who confirmed that the ambiguities we highlighted, if misinterpreted, could open the protocol to attacks such as those we found – and that these attacks will not work if the ambiguous sections are interpreted correctly. Out of concern that a real implementation might have misinterpreted the RFC, we manually analyzed the source for both the Linux and FreeBSD implementations, and tested both implementations with PACKETDRILL, finding that neither made either mistake.

6 Related Work

There are many automated attack discovery tools, each crafted to a particular variety of bug or mechanism of attack, e.g., SNAKE [33] (which fuzzes network protocols), TCPWN [32]

(which finds throughput attacks against TCP congestion control implementations), TAMARIN [44] and PROVERIF [13] (which find attacks against secrecy in cryptographic protocols), KORG [67] (which finds communication attacks against network protocols), and so on. Some of these tools (e.g., KORG) are general purpose, designed to attack any correctness property within a generic specification language, while others (e.g., TAMARIN or PROVERIF) are more specialized, designed to target specific types of properties such as secrecy and trace-equivalence. One work, which studied TCP and ABP, suggested reactive controller synthesis (RCS) as an alternative to KORG's approach [41]. KORG generates attacks that sometimes succeed, depending on choices made by the peers, whereas the RCS method only outputs attacks that always succeed; but such attacks do not always exist. Another approach, which Fiterau-Brostean et. al. [21] successfully applied to various SSH [71] and DTLS [51] implementations, describes incorrect behaviors using automata (rather than properties). This specification style makes sense when generic bug patterns are known ahead of time.

Formal methods such as theorem proving, model checking, property-based testing, and attack synthesis for protocols have been applied to TLS [17] and accountable proxying over it [12], QUIC [43], Bluetooth [69], 5G [9] and its key-establishment stack [45], TCP congestion control [6] and the combination of Karn's Algorithm and the RTO computation upon which it relies [66], the TCP establishment routine [46, 47, 67], and contactless EMV payments [10, 48], to name a few. Compared to many of these systems, such as TCP which has been studied for over 30 years [11, 25, 27, 32, 33, 47, 54, 67], much less is known about the security of SCTP, particularly from an FM perspective.

Of the prior works that applied formal methods to the security of SCTP, only the Uppaal analysis by Saini and Fehnker [53] used a technology (model-checking) that can verify arbitrary properties. They reported two properties in their paper; the first is similar to our ϕ_2. The second says an adversary only capable of sending `INIT` packets cannot cause a victim peer to change state. This property is trivial for us because we use an FSM model where the peer states are precisely the model states. And in our model, the only transition out of Closed that happens upon receiving an `INIT` is a self-loop that sends an `INIT_ACK` and returns to Closed. In contrast, in Saini and Fehnker's model the peer state is a variable in memory, while the model states are totally different (e.g., LC1, LC2). Thus, the property merits verification in their model but not ours. Saini and Fehnker's work is the only one we are aware of that studied SCTP in the context of an attacker using formal methods. But their attacker was only capable of sending `INIT` messages, in contrast to our attacker models which are much more sophisticated, and their attacker could not spoof the port and IP of a peer. Hence, they could not model (and so did not find) the CVE attack.

Another line of inquiry aims to model the performance of SCTP, e.g., using numerical analyses and simulations [16]. For example, Fu and Atiquzzaman built an analytical model of SCTP congestion control, including *multihoming*, an SCTP feature not available in TCP. They compared their model to simulations and found it to be accurate in estimating steady-state throughput of multihomed paths [23]. Such models are also used to evaluate new features, e.g., as in [72].

7 Conclusion

In this work we formally modeled SCTP and specified ten novel LTL correctness properties based on a close reading of the RFCs. We proved that in the absence of an attacker, the protocol satisfies all ten properties. We used KORG to synthesize attacks against our model for four novel attacker models, Off-Path, Evil-Server, Replay, and On-Path, and for two configurations of the SCTP model – one without the RFC 9260 patch and another with it. This required improvements to KORG, which we open-sourced with the paper artifacts. Without the patch, we found the CVE-2021-3772 attack in the Off-Path attacker model; a variety of Evil-Server and On-Path attacks; and one Replay attack. Then we repeated our analysis with the patch, and found that it eliminated the CVE vulnerability. We found that the patch did not impact the security of the protocol with respect to properties other than the one violated by the CVE attack in the Off-Path attacker model, nor did it change any of the results in the other (less realistic) attacker models. Recall, however, that our analysis is limited to DoS attacks involving just two agents in an association, because KORG does not scale to attacker models with very large (let alone unbounded) state-spaces.

We also explored extending KORG to not just discover vulnerabilities, but synthesize patches too. We found the task infeasible as the search space for edits is enormous, and each edit requires re-verifying. And since PROMELA does composition over FIFO channels, reasoning about the composite Kripke Structure and tying it back to the PROMELA encoding proved very challenging. Though we failed to synthesize patches in this work, we believe patch synthesis may be plausible in a more automata-theoretic context.

Our attacks highlight the need to explicitly handle unexpected but valid packets and set reasonable timer values. We reported two ambiguities in RFC 9260, each of which could be misinterpreted in a way that could lead to a vulnerability. We analyzed the Linux and FreeBSD SCTP implementations using PACKETDRILL and found both correctly interpreted the ambiguous portions of text. We concluded with recommendations for how the portions could be made unambiguous in errata. So far, our second recommendation for an erratum was accepted by the SCTP RFC committee.

References

[1] packetdrill. https://github.com/nplab/packetdrill/tree/master. Commit 7f3daabd7feed2b18b958e870f973fec92879d98, accessed 31 July 2023.

[2] SCTP. https://github.com/torvalds/linux/tree/master/net/sctp. Accessed 15 March 2023.

[3] User Datagram Protocol. RFC 768, Aug. 1980.

[4] Sctp. https://man.freebsd.org/cgi/man.cgi?query=sctp&sektion=4&manpath=FreeBSD+7.0-RELEASE, 2006. Accessed 1 May 2023.

[5] Data communication. https://webrtcforthecurious.com/docs/07-data-communication/, November 2022. Accessed 31 July 2023.

[6] ARUN, V., ARASHLOO, M. T., SAEED, A., ALIZADEH, M., AND BALAKRISHNAN, H. Toward formally verifying congestion control behavior. In *SIGCOMM* (2021).

[7] BACKES, M., DREIER, J., KREMER, S., AND KÜNNEMANN, R. A novel approach for reasoning about liveness in cryptographic protocols and its application to fair exchange. In *EuroS&P* (2017), IEEE.

[8] BASIN, D., CREMERS, C., DREIER, J., AND SASSE, R. Symbolically analyzing security protocols using tamarin. *ACM SIGLOG News* (2017).

[9] BASIN, D., DREIER, J., HIRSCHI, L., RADOMIROVIC, S., SASSE, R., AND STETTLER, V. A formal analysis of 5g authentication. In *SIGSAC* (2018), ACM.

[10] BASIN, D., SASSE, R., AND TORO-POZO, J. The emv standard: Break, fix, verify. In *S&P* (2021), IEEE.

[11] BELLOVIN, S. M. Security problems in the tcp/ip protocol suite. *SIGCOMM Computer Communication Review* (1989).

[12] BHARGAVAN, K., BOUREANU, I., DELIGNAT-LAVAUD, A., FOUQUE, P.-A., AND ONETE, C. A formal treatment of accountable proxying over TLS. In *S&P* (2018), IEEE.

[13] BLANCHET, B., ET AL. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends® in Privacy and Security* (2016).

[14] CARDWELL, N., CHENG, Y., BRAKMO, L., MATHIS, M., RAGHAVAN, B., DUKKIPATI, N., CHU, H.-K. J., TERZIS, A., AND HERBERT, T. PACKETDRILL: Scriptable network stack testing, from sockets to packets. In *USENIX ATC* (2013).

[15] CARO, A. L., POON, K., TÜXEN, M., STEWART, R. R., AND ARIAS-RODRIGUEZ, I. Stream Control Transmission Protocol (SCTP) Specification Errata and Issues. RFC 4460, Apr. 2006.

[16] CHUKARIN, A., AND PERSHAKOV, N. Performance evaluation of the stream control transmission protocol. In *MELECON* (2006), IEEE.

[17] CREMERS, C., HORVAT, M., HOYLAND, J., SCOTT, S., AND VAN DER MERWE, T. A comprehensive symbolic analysis of TLS 1.3. In *SIGSAC* (2017), ACM.

[18] DEWOPRABOWO, R., ARZAKI, M., AND RUSMAWATI, Y. Formal verification of divide and conquer key distribution protocol using proverif and tla+. In *ICACSIS* (2018), IEEE.

[19] DOLEV, D., AND YAO, A. On the security of public key protocols. *IEEE Transactions on information theory* (1983).

[20] EDDY, W. Transmission Control Protocol (TCP). RFC 9293, Aug. 2022.

[21] FITERAU-BROSTEAN, P., JONSSON, B., SAGONAS, K., AND TÅQUIST, F. Automata-based automated detection of state machine bugs in protocol implementations. In *NDSS* (2022).

[22] FLOYD, S., HANDLEY, M. J., AND KOHLER, E. Datagram Congestion Control Protocol (DCCP). RFC 4340.

[23] FU, S., AND ATIQUZZAMAN, M. Performance modeling of SCTP multihoming. In *GLOBECOM* (2005), IEEE.

[24] GOGA, N., COSTACHE, S., AND MOLDOVEANU, F. A formal analysis of iso/ieee p11073-20601 standard of medical device communication. In *Systems Conference* (2009), IEEE.

[25] GONT, F. ICMP attacks against TCP. https://datatracker.ietf.org/doc/rfc5927/, july 2022.

[26] GUO, C., AND TÜEXEN, M. Questions on rfc4960 abort init tag equal 0 in init msg and manditory info less than 20 bytes. https://mailarchive.ietf.org/arch/msg/tsvwg/nna1IVrRIKPBKOwmv5JC3X5UQSA/. Accessed 5 February 2024.

[27] HARRIS, B., AND HUNT, R. Tcp/ip security threats and attack methods. *Computer communications* (1999).

[28] HOLZMANN, G. J. The model checker SPIN. *IEEE Transactions on software engineering 23*, 5 (1997).

[29] HOLZMANN, G. J. Redundant software (and hardware) ensured curiosity reached its destination and functioned as its designers intended. *Communications of the ACM* (2 2014).

[30] HOLZMANN, G. J., AND SMITH, M. H. Automating software feature verification. *Bell Labs Technical Journal 5*, 2 (2000).

[31] IYENGAR, J., AND THOMSON, M. QUIC: A UDP-based multiplexed and secure transport. https://datatracker.ietf.org/doc/rfc9000/, May 2021. Accessed 17 May 2024.

[32] JERO, S., HOQUE, M. E., CHOFFNES, D. R., MISLOVE, A., AND NITA-ROTARU, C. Automated attack discovery in tcp congestion control using a model-guided approach. In *NDSS* (2018).

[33] JERO, S., PACHECO, M. L., GOLDWASSER, D., AND NITA-ROTARU, C. Leveraging textual specifications for grammar-based fuzzing of network protocols. In *AAAI Conference on Artificial Intelligence* (2019).

[34] KACER, M., AND LANGLOIS, P. SS7 attacker heaven turns into riot: How to make nation-state and intelligence attackers' lives much harder on mobile networks. *Black Hat* (2017).

[35] KHOT, I. A smaller, faster video calling library for our apps. https://engineering.fb.com/2020/12/21/video-engineering/rsys/, december 2020. Accessed 31 July 2023.

[36] KOBEISSI, N., BHARGAVAN, K., AND BLANCHET, B. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *EuroS&P* (2017), IEEE.

[37] LOHR, H., AND DKNAPPETTMSFT. What's new in the remote desktop WebRTC redirector service. https://learn.microsoft.com/en-us/azure/virtual-desktop/whats-new-webrtc, April 2023. Accessed 31 July 2023.

[38] LONG, X. SCTP enhancements for the verification tag. https://github.com/torvalds/linux/commit/32f8807a48ae55be0e76880cfe8607a18b5bb0df. Accessed 5 February 2024.

[39] LONG, X. https://github.com/torvalds/linux/commit/32f8807a48ae55be0e76880cfe8607a18b5bb0df, October 2021.

[40] MARTINS, M. G. M., ET AL. Modelagem e análise formal de algumas funcionalidades de um protocolo de transporte através das redes de petri. Accessed 2 August 2023 at https://docplayer.com.br/146114380-Modelagem-e-analise-formal-de-algumas-funcionalidades-de-um-protocolo-de-transporte-atraves-das-redes-de-petri.html.

[41] MATSUI, S., AND LAFORTUNE, S. Synthesis of winning attacks on communication protocols using supervisory control theory: two case studies. *Discrete Event Dynamic Systems* (2022).

[42] MATTSSON, J. P. https://datatracker.ietf.org/meeting/115/materials/slides-115-tsvwg-sctp-auth-security-issues-00, 2022.

[43] MCMILLAN, K. L., AND ZUCK, L. D. Formal specification and testing of QUIC. In *SIGCOMM*. 2019.

[44] MEIER, S., SCHMIDT, B., CREMERS, C., AND BASIN, D. The tamarin prover for the symbolic analysis of security protocols. In *CAV* (2013), Springer.

[45] MILLER, R., BOUREANU, I., WESEMEYER, S., AND NEWTON, C. J. The 5g key-establishment stack: In-depth formal verification and experimentation. In *Asia CCS* (2022).

[46] MUSUVATHI, M., ENGLER, D. R., ET AL. Model checking large network protocol implementations. In *NSDI* (2004).

[47] PACHECO, M. L., VON HIPPEL, M., WEINTRAUB, B., GOLDWASSER, D., AND NITA-ROTARU, C. Automated attack synthesis by extracting finite state machines from protocol specification documents. In *S&P* (2022), IEEE.

[48] RADU, A.-I., CHOTHIA, T., NEWTON, C. J., BOUREANU, I., AND CHEN, L. Practical emv relay protection. In *S&P* (2022), IEEE.

[49] RED HAT, I. CVE-2021-3772 detail. https://nvd.nist.gov/vuln/detail/CVE-2021-3772. Accessed 15 March 2023.

[50] RESCORLA, E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, Aug. 2018.

[51] RESCORLA, E., TSCHOFENIG, H., AND MODADUGU, N. The datagram transport layer security (DTLS) protocol version 1.3. https://www.rfc-editor.org/rfc/rfc9147, April 2022.

[52] RÜNGELER, I., AND TÜXEN, M. Sctp support in the inet framework and its analysis in the wireshark packet analyzer. In *Multihomed Communication with SCTP (Stream Control Transmission Protocol)*. CRC Press, 2012.

[53] SAINI, S., AND FEHNKER, A. Evaluating the stream control transmission protocol using Uppaal. *arXiv preprint arXiv:1703.06568* (2017).

[54] SCHUBA, C. L., KRSUL, I. V., KUHN, M. G., SPAFFORD, E. H., SUNDARAM, A., AND ZAMBONI, D. Analysis of a denial of service attack on tcp. In *S&P* (1997), IEEE.

[55] STEWART, R. Stream control transmission protocol. https://www.rfc-editor.org/rfc/rfc4960, September 2007. Accessed 23 February 2023.

[56] STEWART, R., TUEXEN, M., AND CAMARILLO, G. Security attacks found against the stream control transmission protocol (SCTP) and current countermeasures. https://datatracker.ietf.org/doc/html/rfc5062, 2007.

[57] STEWART, R., TÜXEN, M., AND LEI, P. SCTP: What is it, and how to use it? In *BSDCan* (2008).

[58] STEWART, R., TÜXEN, M., AND NIELSEN, K. Stream control transmission protocol. https://www.rfc-editor.org/rfc/rfc9260, June 2022. Accessed 15 March 2023.

[59] STEWART, R., XIE, Q., MORNEAULT, K., SHARP, C., SCHWARZBAUER, H., TAYLOR, T., RYTINA, I., KALLA, M., ZHANG, L., AND PAXSON, V. Stream control transmission protocol. https://www.rfc-editor.org/rfc/rfc2960, October 2000. Accessed 15 March 2023.

[60] THOMSEN, S. E., AND SPITTERS, B. Formalizing nakamoto-style proof of stake. In *CSF* (2021), IEEE.

[61] TÜXEN, M. Rfc 9260 errata. https://www.rfc-editor.org/errata_search.php?rfc=9260, 4 2024.

[62] VANIT-ANUNCHAI, S. Towards formal modelling and analysis of SCTP connection management. In *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools* (2008).

[63] VANIT-ANUNCHAI, S. Validating SCTP simultaneous open procedure. In *FSEN* (2013), Springer.

[64] VARDI, M. Y., AND WOLPER, P. An automata-theoretic approach to automatic program verification. In *LICS* (1986), IEEE Computer Society.

[65] VASS, J. How Discord handles two and half million concurrent voice users using WebRTC. https://discord.com/blog/how-discord-handles-two-and-half-million-concurrent-voice-users-using-webrtc, September 2018. Accessed 31 July 2023.

[66] VON HIPPEL, M., MCMILLAN, K. L., NITA-ROTARU, C., AND ZUCK, L. D. A formal analysis of karn's algorithm. In *NETYS* (2023), Springer.

[67] VON HIPPEL, M., VICK, C., TRIPAKIS, S., AND NITA-ROTARU, C. Automated attacker synthesis for distributed protocols. In *SAFECOMP* (2020), Springer.

[68] WANG, J., ZHANG, S., AND CHEN, F. Modeling and verification of sctp association management based on colored petri nets. In *ISECS International Colloquium on Computing, Communication, Control, and Management* (2008), IEEE.

[69] WU, J., WU, R., XU, D., TIAN, D. J., AND BIANCHI, A. Formal model-driven discovery of bluetooth protocol design vulnerabilities. In *S&P* (2022), IEEE.

[70] YAO, J., TAO, R., GU, R., AND NIEH, J. Mostly automated verification of liveness properties for distributed protocols with ranking functions. In *POPL* (2024).

[71] YLONEN, T., AND LONVICK, C. The secure shell (SSH) transport layer protocol. https://www.rfc-editor.org/rfc/rfc4253, january 2006.

[72] ZOU, J., UYAR, M. Ü., FECKO, M. A., AND SAMTANI, S. Throughput models for SCTP with parallel subflows. *Computer Networks* (2006).

8 Appendix

History of CVE-2021-3772. The vuln. reported in CVE-2021-3772 arose from the following text in RFC 4960 [55]: "If the value of the Initiate Tag in a received `INIT` chunk is found to be 0, the receiver MUST treat it as an error and close the association by transmitting an `ABORT`." As shown in Fig. 3, if an implementation did not check the validity of the `INIT` before transmitting an `ABORT`, then the RFC allowed for a DoS where the attacker would transmit an invalid `INIT` and trigger the victim to close an otherwise valid association. This vuln. was first reported in the SCTP mailing list [26] and then in CVE-2021-3772 [49]. The vuln. was subsequently patched in Linux by swapping the order of operations, to ensure the vtag is always checked before the itag [38]. The RFC was updated in 9260 [58] to say: "If the value of the Initiate Tag in a received `INIT` chunk is found to be 0, the receiver MUST silently discard the packet." ... and FreeBSD [4] implemented this patch when it was updated to reflect the new RFC.

User Model. The user nondeterministically sends User_Assoc, User_Abort, and User_Shutdown.

Additional Errata. We suggest incorporating the self-loop at Closed that occurs upon receiving a `SHUTDOWN_COMPLETE`, into the State Association Diagram in Section 4. We suggest incorporating "`INIT` collision" into that same diagramWe suggest expanding 5.2 to explain how to handle other unexpected chunks, e.g., `COOKIE_ERROR`, `SHUTDOWN_COMPLETE`, etc.

Performance We time our experiments and patch verification tasks, and almost always, KORG terminates in seconds

```
/* P1 */ G((st[0] == Closed) -> (X(F(st[0] == Closed
|| st[0] == Established || st[0] == CookieWait))))
/* P2 */ G(F(st[0] != ost[0] || st[1] != ost[1] ||
(st[0] == Closed && st[1] == Closed) ||
(st[0] == Established && st[1] == Established)))
/* P3 */ G((st[0] != ost[0] && ost[0] ==
ShutdownAckSent) -> (st[0] == Closed))
/* P4 */ G(F(st[0] != CookieEchoed ||
timers[0] == T1_COOKIE))
/* P5 */ G(st[0] != ShutdownReceived ||
st[1] != ShutdownReceived)
/* P6 */ G((st[0] != ost[0] && ost[0] ==
ShutdownReceived) -> (st[0] == ShutdownAckSent ||
st[0] == Closed))
/* P7 */ G(st[0] != CookieEchoed || st[1] !=
ShutdownReceived)
/* P8 */ G((ost[1] == Established && ost[0] ==
Closed && everAborted == false && everTimedOut ==
false && ost[0] != st[0]) -> (st[0] == Established
|| st[0] == IntermediaryCookieWait))
/* P9 */ G((ost[0] == Established && ost[1] ==
Closed && everAborted == false && everTimedOut ==
false && ost[1] != st[1]) -> (st[1] == Established
|| st[1] == IntermediaryCookieWait))
/* P10 */ G((ost[0] == Established && (st[0] ==
ShutdownSent || st[0] == ShutdownReceived))
-> F(st[0] == Closed))
```

Figure 13: Our 10 LTL properties are formulated in PROMELA. We define our atomic propositions as follows in PROMELA, where `st` holds the state of each peer, `ost` holds the prior one, and `timers` holds the peers' timers.

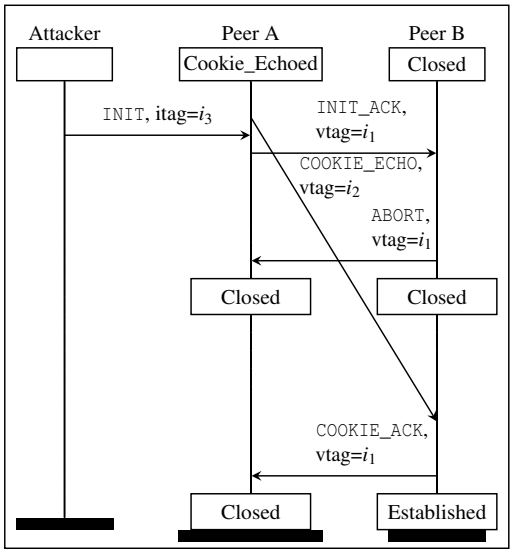

Figure 14: First ambiguity attack.

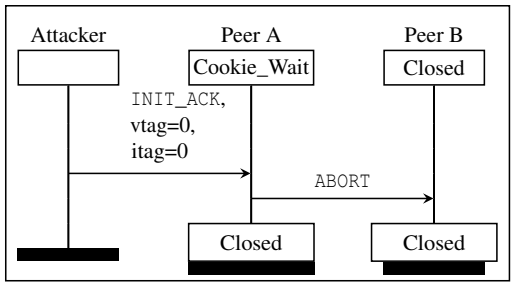

Figure 15: Second ambiguity attack.

```
chan attacker_mem = [2] of {
    mtype:msgs,mtype:tag,mtype:tag,byte };
active proctype attacker_replay() {
  mtype:msgs b_0; mtype:tag b_1, b_2; byte b_3;
  do :: atomic { AtoB ?? <b_0, b_1, b_2, b_3>
    -> attacker_mem ! b_0, b_1, b_2, b_3; }
  :: atomic { attacker_mem ?? b_0, b_1, b_2, b_3
    -> AtoB ! b_0, b_1, b_2, b_3; }
  :: atomic { attacker_mem ?? b_0, b_1, b_2, b_3; }
  :: break; od }
```

Figure 16: The replay attacker gadget.

or minutes, with one interesting exception. In the Off-Path experiments, KORG takes about two hours to confirm that no attacks exist against ϕ_8, and about 1.5hrs to find the CVE attack against ϕ_9. Recall that ϕ_8 and ϕ_9 are identical, except that the peer roles are reversed. Further inspection reveals these two properties are the largest in our property set, and the Off-Path attacker model is the largest attacker model. The reason these two analyses take longer follows, as KORG reduces to LTL model-checking, the runtime of which is polynomial in the size of the model and $O(\log^2|\phi|)$ in the size of ϕ [64].

	Off-Path		Evil-Server		Replay		On-Path	
	E	P	E	P	E	P	E	P
ϕ_1	2:20	2:13	0:23	0:23	0:3	0:3	0:15	0:15
ϕ_2	8:43	11:14	0:21	0:21	0:2	0:2	0:26	0:26
ϕ_3	3:20	12:53	0:20	0:20	0:2	0:2	0:25	0:25
ϕ_4	1:45	1:26	0:11	0:11	0:2	0:2	0:14	0:14
ϕ_5	2:57	1:35	0:10	0:10	0:2	0:2	0:12	0:12
ϕ_6	3:19	18:8	0:20	0:20	0:2	0:2	0:25	0:25
ϕ_7	1:43	4:41	0:11	0:10	0:2	0:2	0:13	0:14
ϕ_8	123:42	7:7	1:6	1:7	0:2	0:2	1:34	1:34
ϕ_9	86:10	6:48	1:5	1:5	0:2	0:2	0:11	0:11
ϕ_{10}	0:4	0:4	0:3	0:4	0:2	0:2	0:4	0:4

Table 3: Time taken (min:sec) to perform each (E) experiment and (P) patch verification on a 16GB M1 Macbook Air.