On the Soundness of Infrastructure Adversaries

Alexander Dax and Robert Künnemann
CISPA Helmholtz Center for Information Security
Saarland Informatics Campus

***Abstract*—Companies and network operators perform risk assessment to inform policy-making, guide infrastructure investments or to comply with security standards such as ISO 27001. Due to the size and complexity of these networks, risk assessment techniques such as attack graphs or trees describe the attacker with a finite set of rules. This characterization of the attacker can easily miss attack vectors or overstate them, potentially leading to incorrect risk estimation.**

In this work, we propose the first methodology to justify a rule-based attacker model. Conceptually, we add another layer of abstraction on top of the symbolic model of cryptography, which reasons about protocols and abstracts cryptographic primitives. This new layer reasons about Internet-scale networks and abstracts protocols.

We show, in general, how the soundness and completeness of a rule-based model can be ensured by verifying trace properties, linking soundness to safety properties and completeness to liveness properties. We then demonstrate the approach for a recently proposed threat model that quantifies the confidentiality of email communication on the Internet, including DNS, DNSSEC, and SMTP. Using off-the-shelf protocol verification tools, we discover two flaws in their threat model. After fixing them, we show that it provides symbolic soundness.

***keywords*—protocol verification; planning; security economics; DNSSEC; DNS**

I. Introduction

The Internet is the primary medium for distributing entertainment, news and knowledge and an important pillar to industrial commerce. It is constructed from service providers interoperating according to several protocols. Many of them were conceived before the Internet was even considered a Mass Medium [43]; they were hence designed to be fast and service-oriented, whereas security was a second thought. Trust between service providers at different protocol layers is thus an implicit assumption, making it difficult to estimate potential attacks' impact.

High-profile attacks, e.g. on routing [39] or name resolution [44] are a painful reminder of these trust assumptions. They also highlight the slow adoption of security protocols, which were developed only post-hoc, to mitigate some of these issues. Even for TLS [48], which enjoys high popularity, adoption was and remains slow. According to Qualys Labs [35], 6% of all websites still support SSL 3.0, which is exploitable in various manners and was deprecated in 2015. Moreover, security protocols rely on trust assumptions and a complicated interplay between routing, name resolution, and the application layer. An example is RFC 7817 [42], which defines certificate validation for email transport. It mandates that the certificate contains the email domain (the part after the '@') and not just the target server's domain name, as a name resolution attacker can easily manipulate the latter. Large-scale attacks thus rarely exploit previously unknown flaws in a single protocol, but instead target their deployment in the wild.

Despite the effort put into securing individual protocols and cryptographic primitives in the past decades, worldwide attacks like the Great Cannon [40] or spying systems like PRISM [26] exploit weak components and (the absence of) trust anchors in the infrastructure. To analyze an infrastructure like the Internet, with broken legacy protocols, unstable trust assumptions, and varying degrees of centralization on different layers, a high-level approach is necessary.

Risk assessment originates in the formal assessment of potential failures in large infrastructures like power plants. Techniques like fault trees provide a systematic method for identifying and minimizing potential risks. They were soon adopted for IT infrastructure. These techniques usually consider the severity of known vulnerabilities and some valuation of critical assets. The problem size grows with the size of the network. Therefore, most of these techniques formalize the threat model as a set of rules. Those techniques include planning, attack graphs (which were derived from fault graphs), and game-based models.[1]

While these analyses are formal and well justified, the rules themselves are not formally justified. It is not safe to assume that the set of rules is comprehensive. Thus the analysis may miss potential attack vectors. There is a surprising similarity to the soundness of the Dolev-Yao model. Abadi and Rogaway's seminal paper on computational soundness [2] considered the soundness of a such a rule-based symbolic attacker on protocols in the computational model. Likewise, our focus is on the soundness of a rule-based attacker, the infrastructure attacker (IA), but in the symbolic model instead of the computational model. In both cases, the need for further abstraction is driven by the complexity of the problem (infrastructure analysis/protocol analysis) but requires justification.

Contributions

1) We define proof obligations for the correctness of an infrastructure attacker in the STRIPS framework for planning as a set of trace properties. We show that soundness can be proven by verifying *safety properties*, and correctness by verifying *liveness properties*.

[1] For other techniques, consider the study by Wang et al. [55].

2) We apply this definition to an IA model for email communication [52] and establish its soundness (barring some minor flaws).
3) We show how to automate the proof of this trace properties by over-approximating all possible instantiations of the IA model with a single process. The protocol transformations we introduce to this end are of independent interest, as they can help to reduce drastically the size of processes that model an adversary with limited access to network traffic.
4) We show various authentication properties of SMTP in conjunction with DNS, DNSSEC, and a simple resolver model in ProVerif. As a by-product, this model provides the first automated verification result for authentication in DNSSEC.

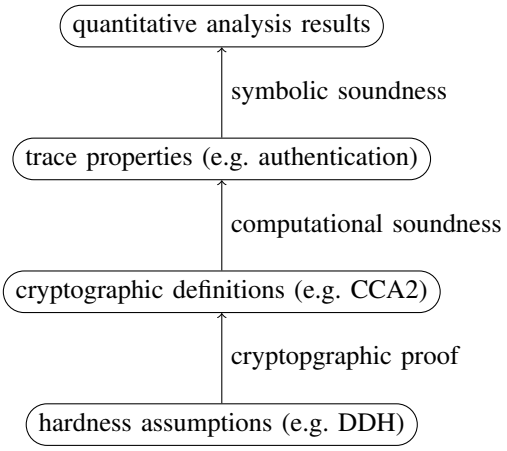

Fig. 1: Relation between different levels of abstraction

II. Related Work

1) Risk assessment techniques: The most popular techniques for the analysis of IT infrastructure are attack graphs [56] and trees[49], see [38] for a recent survey. They originate in risk assessment and reliability analysis for critical infrastructures. Fault tree analysis [16] was used, e.g. for analyzing nuclear power plants or military missile control systems. Attack graphs and trees have been used to assess risks in forensic examination [37], network security [34, 47] or cloud infrastructures [3]. Used naïvely, both techniques suffer from the state explosion property. Luckily, a large body of work is devoted to improving performance, e.g. generating of minimal attack graphs [25], distributing attack graph generation [32] or the efficient representation of network defenses [30].

More recently, planning was considered as an alternative technique with great benefits in terms of performance [24]. Planning is one of the oldest sub-areas of AI and benefits from being a well-studied research field with a large community and a focus on optimizing performance. Compared to various semantics for attack graphs, there is a fairly wide agreement on the STRIPS framework [22]. Planning was used for attack graph generation [27], network analysis [12], penetration testing [45], and internet infrastructure analysis [53]. Additionally, the popular attack graph formalism can be translated into a planning problem [29].

2) Infrastructure analysis: Until recently, these approaches were used to analyze local networks or the public infrastructure unrelated to information security. Presumably, this was due to the problem size associated with large-scale infrastructures like the Internet. Frey et al. [23] conducted one of the first Internet-scale infrastructure assessments in terms of security evaluation. They investigated the Border Gateway Protocol deployment looking into potential threats and vulnerabilities. Simeonovski et al. [51] present a technique that models services, providers, and dependencies on the Internet as a property graph, establishing a high-level IA model. This model is used to reason about dependencies between services and infrastructure providers and how these dependencies can be exploited to impact a large amount of end users. They conduct a large-scale case study by using a simple tainting-style propagation technique in a graph database highly optimized for reachability queries. They studied several attack scenarios like email-sniffing and DDOS caused by the distribution of malicious JavaScript. More recently, Speicher et al. [52] introduced the first deployment analysis on a global level, evaluating various measures to secure the email infrastructure against large-scale attacks. They employ Stackelberg planning [53], which is a two-stage planning technique that computes all defender plans that are Pareto-optimal with respect to their cost and the worst-case impact of an attacker.

To our knowledge, all these assessment approaches have only informally justified their threat models. Given the high abstraction level of their reasoning, validation using formal analysis techniques is necessary. Sheyner et al. [50] propose the use of symbolic model checking to generate attack graphs from a finite state machine that represents the network. Here, state transitions correspond to atomic attacker steps which themselves require justification. This approach is neither applicable to larger networks (because of the aforementioned state-explosion property) nor does it provide the desired level of justification (as network attackers are too complex for finite state machines).

3) The analogy to computational soundness: In contrast to cryptographic primitives like encryption or signatures, larger cryptographic protocols are typically analyzed in the Dolev-Yao model [20], where cryptographic primitives (short: crypto primitives) are abstracted with a term algebra. Proofs in the computational model are possible, but become prohibitively complex due to the need to reason about probabilistic behavior and runtime. Mechanization becomes incredible difficult [8] and manual proofs can easily miss details. By contrast, the abstraction of cryptography by a term algebra has enabled the development of fully automatic and semi-automatic protocol verifiers [10, 41] that can handle highly complex protocols, e.g. TLS 1.3 [9, 18].

To bridge both worlds, Abadi and Rogaway [2] introduced *computational soundness*, justifying the use of the symbolic model for protocol analysis. Gaining the advantages of automation in the symbolic model, and the stronger guarantees of the

	computational sound.	symbolic soundness
threat model	network attacker	infrastructure att.
assumption	perfect cryptography	perfect protocol
high-level semantics	term algebra + process calculus	planning (STRIPS)
low-level semantics	probabilistic Turing Machines	term algebra + process calculus
Proof strategy		
Fix a set of	crypto primitives	protocols
conforming to	crypto definitions (e.g. CCA2 for Cramer-Shoup).	trace properties (e.g. authenticity for TLS).
For all	protocols	network topologies
map each	computational traces	protocol trace
from	comp. executions (interpreted by TM)	processes (compiled from the topology)
to a	(symb.) protocol trace.	plan.

TABLE I: Comparing computational soundness and symbolic soundness.

computational model, the notion of computational soundness is seen as a massive milestone in protocol verification. With this work, we want to extend on this stack in an analogous fashion, and introduce the notion of *symbolic soundness* relating the infrastructure adversary to the symbolic model in a similar fashion. This is depicted in Fig. 1.

To attain this goal, we lift this approach from the protocol level to the infrastructure level. To readers familiar with computational soundness, Tab. I can help to support this analogy.

We formalize the IA model as a planning problem in the STRIPS formalism [22]. The rules represent an infrastructure attacker that can selectively corrupt parts of the infrastructure, but assumes that protocols themselves are secure. Dolev-Yao models, by contrast, represent a network attacker, but assume the crypto primitives to be perfect. To reason about the validity of the model w.r.t. those assumptions, they need to be formalized. As they are implicit in the respective semantics (STRIPS / process calculus with term algebra), a low-level semantics is necessary to state these assumptions. Assumptions about protocols are stated in the Dolev-Yao model, usually within a process calculus with a term algebra. Assumptions about cryptographic primitives are stated as asymptotic probability bounds on the probability of a runtime-bounded Turing machines winning some game.

Symbolic soundness asserts that, when compiling a given infrastructure model into a process, all symbolic traces that this process allows can be mapped to attacker plans in the planning model. This is structurally similar to computational soundness, which ensures that no computational attacks are missed by mapping all computational executions to symbolic traces (with a negligible failure probability). We can thus show that no symbolic attack is missed by the IA model, provided the protocols are working as intended. To be clear: symbolic soundness does not imply computational soundness, but both combine. If a symbolic soundness result asserts the absence of attacks w.r.t. to some symbolic model, then a computational soundness can extend this result to the computational model of cryptography. This requires that the computational soundness result supports the cryptographic primitives and process calculus used by the symbolic soundness result.

Much like results in computational soundness, symbolic soundness results apply to a fixed set of primitives, in their case: protocols. We model an infrastructure consisting of DNS, DNSSEC and SMTP, using a dialect of the applied-π calculus [1].

4) Symbolic completeness and liveness properties: In contrast to computational soundness, the completeness of an IA model is equally important to the analysis. A quantitative analysis, e.g. counting the number of affected hosts, is incorrect if the IA model overestimates the protocol attacker's capabilities. In the case of Stackelberg planning, this might lead to the proposition of suboptimal countermeasures and, if the defender budget is fixed, to an allocation that is not optimal for security. We therefore define *symbolic completeness* and show a set of conditions that implies the completeness of this model. Unfortunately, one of these is a liveness property, i.e., a property of the form: 'the [protocol] eventually enters a desirable state' [36]. Practically all protocol verification tools [10, 41, 17] in the unbounded model cover only safety properties, i.e., properties of the form 'the protocol never enters a bad state'. Hence, there is currently no support for the verification of liveness properties such as ours. We further elaborate on this topic in section IV-C.

5) Analysis of DNSSEC: To our knowledge, our case study provides not only the first automated result w.r.t. an infrastructure attacker, but also the first automated verification result for DNSSEC. Chetioui et al. [14] investigate (weak) secrecy in E-DNSSEC, a variant of DNSSEC that adds encryption, in ProVerif. Kammuller [31] also cover authentication in a handwritten, but automatically verified proof in Isabelle/HOL.

III. Background: Automated Planning

A planning task is usually described in the STRIPS framework [22]. Here, $\Pi = (\mathcal{P}, \mathcal{I}, \mathcal{A}, \mathcal{G})$ is defined over a high-level representation of the world in which each state σ is built over a set of *state propositions* $\mathcal{P}$. $\mathcal{I} \subseteq \mathcal{P}$ is the initial state and the task is to reach a *goal states* in $\mathcal{G} \subseteq 2^{\mathcal{P}}$. A set of *actions* $\mathcal{A}$ over $\mathcal{P}$ defines transitions between states. Actions are described as a triple (*pre, del, add*) where *pre* $\subseteq \mathcal{P}$ is the set of preconditions needed in the current state to make the action applicable, *del* $\subseteq \mathcal{P}$ tells which proposition will be deleted in the transition to the next state whereas *add* $\subseteq \mathcal{P}$ tells which propositions are added. In *classical planning*, we assume that all actions have a deterministic effect and that the initial state of the world is known from beginning. A state σ is *reachable* from $\mathcal{I}$, if there is a sequence of actions $a_0 \cdots a_k$, which can be applied to $\mathcal{I}$ one after another resulting in σ. We call this sequence of actions a plan π to reach σ. The basic idea behind planning is to find a sequence of actions, s.t. their application starting from the initial state $\mathcal{I}$ leads to one of

the goal states in $\mathcal{G}$. Speicher et al., e.g., consider the initial state as the nodes that an attacker controls from the start, e.g., different nation-state adversaries or companies abusing power. Goal states are valuable assets that need to be protected, e.g., the largest mail providers within some country. Over the years, several variations of automated planning have been developed, with different modeling assumption and resulting complexity classes for plan existence, worst-case runtime, etc.

We focus on classical planning for the ease of presentation. Our approach easily transfers to probabilistic planning when considering uncertainty about the initial set-up or effect probabilities as model parameters. We cannot justify these parameters via protocol verification (which is typically possibilistic) or cryptographic reasoning in general. These parameters model uncertainty about the attacker's capabilities and intentions. They are thus outside the current scope of formal analysis in security. Our infrastructure attacker is described by actions that have only positive preconditions and postconditions, i.e., they are described as pairs $(pre, post) \in \mathcal{P}^2$ instead of tuples (pre, del, add). Such planning tasks are called *delete-relaxed* or *monotonous* and are easier to solve. Delete-relaxed planning aligns with the implicit assumption that attackers only gain assets in attack graph analysis [5].

Stackelberg planning [53] elevates this form of analysis to a two-player planning task in an attacker/defender scenario. In this scenario, the defender tries to implement mitigation strategies to limit the impact of the worst-case attacker strategy. A Stackelberg planning task differs from a classical task by dividing the set of actions into *leader* (or attacker) actions $\mathcal{A}^{\mathcal{L}}$ and into *follower* (or defender) actions $\mathcal{A}^{\mathcal{F}}$. Further, the goal states are now defined for the defender, namely defender/follower goals $\mathcal{G}^{\mathcal{F}}$. In this setting, an attack is composed of attacker actions, but applies to a world state where the defender has applied a plan composed of defender actions to the initial state. Every attack is annotated with some attacker reward, which depends on the severity of the attack (e.g. number of corrupted connections due to the attack). Defender actions come with a cost. The Stackelberg planning algorithm computes the set of Pareto-optimal pairs of attacker and defender plans. For the soundness of the attacker model, it is enough to consider the classical planning task where the follower actions are removed and only the attacker goal is considered, but the initial state can be any state reachable via defender actions. In our analysis, the initial state is, in fact, arbitrary.

IV. Symbolic Soundness and Completeness

We introduce the concepts of *symbolic soundness* and *symbolic completeness*, which relate the infrastructure adversary model (formalized as a planning problem) to the Dolev-Yao model [20]. Our approach applies to security properties that can be expressed as trace properties. We start by introducing the necessary notation and concepts. Then we introduce the conditions under which symbolic soundness and symbolic completeness hold. Finally, we prove these statements.

A. Notation

For a sequence $s \in \Sigma^*$, let $set(s)$ be the set of elements in s. For $e \in \Sigma$, $s \circ e$ denotes the concatenation with e. For $S \subseteq \Sigma$, $s|_S$ is s with every element outside S removed.

The IA model is formalized in terms of a finite set of planning actions. We define $postseq(\pi) = post_0, post_1, \ldots, post_n$ to be the sequence of postconditions of some plan $\pi = (pre_0, post_0), \ldots, (pre_n, post_n)$. We define a planning trace of some plan π as a sequence $pt = s_1, s_2, \ldots, s_n$, where for all $i \in \{1, \ldots, n\}$, $s_i \in perm(post_i)$ is some permutation of $post_i$. If all postconditions in π are singleton sets, it has only one pt. Let $\mathcal{T}^{\Pi}(\sigma)$ be the union of all planning traces reaching σ, and (with slight abuse of notation) $\mathcal{T}^{\Pi} \subseteq \mathcal{P}^*$ denote the union of planning traces over all states.

For generality, symbolic soundness and completeness are formulated independent of the process calculus. We assume a set of traces of form $traces = Events^*$ that represents the possible behavior of a protocol and is usually specified by encoding it into a process. To simplify the presentation and avoid introducing a mapping function, we assume a non-empty intersection between predicates $\mathcal{P}$ and events $Events$. Our aim is to match planning traces and protocol traces on this intersection, which we denote by $\Sigma_{\cap}$. Typically, the predicates/events in this set signify the corruption of some party or the partial compromise of certain infrastructure services (cf. Table III for examples). We hence call them corruption predicates.

Definition 1. *$\approx$-equivalence*
Let $\approx = (\mathcal{T}^{\Pi} \cup traces)^2$ *s.t.* $s \approx t \iff s|_{\Sigma_{\cap}} = t|_{\Sigma_{\cap}}$.

When all predicates $\mathcal{P}$ are contained in $\Sigma_{\cap}$, our approach can be seen as a refinement, where planning traces provide an abstract view on protocol traces.

B. Symbolic Soundness

We define the symbolic soundness of a planning task w.r.t. a set of traces. We will then provide sufficient conditions for this property. Two of them can be checked statically on the planning problem; the third holds for most process calculi. The fourth induces a set of trace properties that can be discharged to protocol verifiers. We say that a planning task is sound if any behavior of the protocol, e.g. an attack, is represented in the planning task.

Definition 2 (Symbolic Soundness)**.** *A planning task Π is symbolically sound w.r.t. a set of traces $\mathcal{T} \subset Events^*$, if for every trace $t \in \mathcal{T}$, there is a planning trace $pt \in \mathcal{T}^{\Pi}$ s.t. $pt \approx t$.*

Symbolic soundness provides guarantees with respect to the Dolev-Yao model. In case the Dolev-Yao model (represented by $\mathcal{T}$) is covered by a computational soundness result, these guarantees may translate to the computational model, but a priori, these are guarantees in a symbolic model of cryptography. We now state and discuss sufficient conditions for soundness for an arbitrary but fixed planning problem $\Pi = (\mathcal{P}, \mathcal{I}, \mathcal{A}, \mathcal{G})$ and a set of traces $\mathcal{T}$.

CS 1. *All postconditions are singleton.*

$$\forall a = (pre, post) \in \mathcal{A} : |post| = 1$$

Discussion. This condition is true w.l.o.g. for all monotonic planning tasks [13] whose postconditions are positive. Any action with $n > 1$ postconditions $a = (pre, \{c_1, ..., c_n\})$ can be split into n actions $a_i = (pre, \{c_i\})$ without losing completeness or soundness. Since we never delete any information from the state, each plan where a occurs can be recovered by substituting a with the sequence $a_1, \ldots, a_n$. Conversely, we can apply a whenever any plan contains some a_i.

CS 2. *All corruption predicates are reproducible in the planning model.*

$$\forall e \in \Sigma_\cap . \exists (pre, post) \in \mathcal{A} \ : \ post = \{e\}$$

Discussion. This condition is largely technical. Note first that $post$ is singleton by CS 1. The set of corruption predicates $\Sigma_\cap$ should be chosen to represent all events where planning traces and protocol traces ought to match. Hence the planning model must be able to produce them. Furthermore, any planning task can be transformed so that all predicates in $\mathcal{P}$ appear in some action's postcondition without altering the set of reachable states: First, we let $\mathcal{I} = \varnothing$ and add an action that reaches the previous initial state. Now all actions with preconditions that do not appear in any postcondition do not apply and can thus be removed. We then define $\mathcal{P}$ be set to the union of postconditions. As $\Sigma_\cap \subseteq \mathcal{P}$, this implies CS 2.

CS 3. *The set of traces is prefix-closed. For any* $k > 0$

$$\forall e_1, \ldots, e_k . (e_1, \ldots, e_k) \in \mathcal{T} \implies (e_1, \ldots, e_{k-1}) \in \mathcal{T}$$

Discussion. This condition concerns the semantics of the process calculus. It holds for ProVerif [10], Tamarin [41] and Scyther [17].

CS 4. *The production of predicates in* $\Sigma_\cap$ *is* not *dependent on predicates outside of this set.*

$$\forall post \in \Sigma_\cap . \forall (pre, \{post\}) \in \mathcal{A} . \forall f \in pre : f \in \Sigma_\cap$$

Discussion. The corruption predicates $\Sigma_\cap$ are used to describe the security model in both languages. With this condition we restrict the model to independent of predicates outside of $\Sigma_\cap$. We refrain from forbidding predicates outside of $\Sigma_\cap$ in the planning model as they appear to be useful in quantitative tasks. For instance, counting occurrences of specific corruption predicates can be essential in a quantitative analysis. Such a model would be depended on predicates in $\Sigma_\cap$ but not vice versa.

CS 5. *Let* $\mathcal{A} = \mathcal{A}_{c_1} \uplus \mathcal{A}_{c_2} \uplus \cdots \uplus \mathcal{A}_{c_n}$ *be the set of actions, partitioned into disjunct sets* $\mathcal{A}_{c_i}$*, where there is exactly one set per postcondition* $\{c_i\}$*. (By* CS 1*, all postconditions are singleton.) We assume that, whenever a postcondition* c_i *appears in a trace, then a matching precondition appears, too, namely the precondition of* some *action in* $\mathcal{A}_{c_i}$*.*

$$\forall i \in \{1..n\}, t \in \mathcal{T} : c_i \in t \wedge c_i \in \Sigma_\cap \implies \exists a = (pre_i, \{c_i\}) \in \mathcal{A}_{c_i} : \forall g \in pre_i : g \in t.$$

Discussion. This property is a safety property and can be shown using any protocol verifier that handles correspondence properties, e.g. Tamarin [41] or Scyther [17]. In Section VIII, we use ProVerif [10] to this end.

The following theorem establishes the soundness of this approach:

Theorem 1. *If CS 1, CS 2, CS 3, CS 4 and CS 5 hold, then* Π *is symbolically sound.*

Proof. Proof by induction over the length of $t|_{\Sigma_\cap}$.
Base case $|t|_{\Sigma_\cap}| = 0$*:* Let $\sigma = \mathcal{I}$. Then $\mathcal{T}^\Pi(\sigma) = set(())$. For the empty trace t, it holds that $t|_{\Sigma_\cap} = () \in \mathcal{T}^\Pi(\sigma)$.
Inductive step: Let $|t|_{\Sigma_\cap}| = k + 1$.Let $t|_{\Sigma_\cap} = (e_1 e_2 .. e_{k+1})$. By CS 3 and the inductive hypothesis, there is a $t_k|_{\Sigma_\cap} = (e_1 e_2 .. e_k)$ and a planning trace pt_k, with $pt_k \approx t_k$. From pt_k we can infer that there exists a reachable state (of Π) σ_k with $\sigma_k|_{\Sigma_\cap} = \{e_1, e_2, .. e_k\}$.

By CS 1, we get that all postconditions of any action in $\mathcal{A}$ are singleton sets. By CS 2, there exists an action $a \in \mathcal{A}$ with $a = (pre_a, post_a)$ and $post_a = e_{k+1}$. Let $\mathcal{A}_{e_{k+1}}$ be the partition of all of A containing all actions with postcondition e_{k+1}. As $e_{k+1} \in \Sigma_\cap$, by CS 4 all preconditions are in $\Sigma_\cap$, too. By CS 5, there exists an action $a^* = (pre, \{e_{k+1}\})$ s.t. for all $g \in pre : g \in t_k$. As $pre \subseteq \Sigma_\cap$, all $g \in pre$ are in $set(t_k|_{\Sigma_\cap}) = set(pt_k|_{\Sigma_\cap})$ and thus in σ_k. Applying a^* to the state, we get σ_{k+1} with $\sigma_{k+1}|_{\Sigma_\cap} = \{e_1, e_2, .. e_k, e_{k+1}\}$.

Finally, we can conclude that there exists a planning trace $pt_{k+1} \in \mathcal{T}^\Pi(\sigma_{k+1})$ s.t. $t \approx pt_{k+1}$, namely $t \approx e_1 ... e_k e_{k+1} \approx pt_k \circ e_{k+1} = pt_{k+1}$. □

C. Symbolic Completeness

The complementing property to symbolic soundness is symbolic completeness. It ensures that the planning model does not introduce spurious attacks that cannot occur in the protocol model. Planning problems are frequently used to perform a quantitative assessment of, e.g. the number of reachable goal states or the probability of reaching certain assets. The correctness of such an assessment relies on symbolic soundness *and* symbolic completeness. This is in contrast to computational completeness, which is of little interest as long as the symbolic model is good enough to provide verification results.

Definition 3 (Symbolic Completeness)**.** *A planning task* Π *is symbolically complete w.r.t.* $\mathcal{T}$ *if for every planning trace* pt*, there is a trace* $t \in \mathcal{T}$ *s.t.* $pt \approx t$*.*

We provide an additional assumption that ensures symbolic completeness. Unfortunately, it is a *liveness property*, i.e., a property of the form: 'the [protocol] eventually enters a desirable state' [36] and cannot be verified by the current generation of protocol verifiers.

CC 1. *If an action is available and the trace contains the necessary preconditions, then the trace can be extended so it contains this action's postcondition.*

$$\forall t \in \mathcal{T}, a = (\{p_1, ..., p_n\}, c) \in \mathcal{A} : \\ c \in \Sigma_\cap \wedge (\{p_1, ..., p_n\}) \subset set(t) \implies \\ \exists t' \in \mathcal{T} : t' = t \circ t_r \wedge (set(t_r) \cap \Sigma_\cap) = \{c\}.$$

Discussion. Lamport [36] informally describes such properties as liveness properties. Note that here, the 'desirable state' is an additional attack step. As we only consider finite traces, Alpern and Schneider's definition of liveness [4], — which is well known because it decomposes trace properties into safety and liveness properties — does not classify CC 1 as a liveness property.[2] Other characterisations do, see Kindler [33] for a survey.

Nevertheless, state-of-the-art protocol verifiers in the unbounded setting [10, 41, 17] only support the specification of properties of the form $\forall t \in \mathcal{T}.\varphi(t)$ where φ is a property that is protocol-agnostic, i.e. invariant w.r.t. $\mathcal{T}$. This prohibits a direct encoding of CC 1.

Backes, Dreier, Kremer, and Künnemann propose an encoding of liveness properties for Tamarin that allows transforming liveness properties into this fragment of safety properties [7]. Their methodology is based on the idea that the protocol specifies a way to reach the 'desirable state,' e.g. by defining a recovery protocol. Hence any trace either already reached a desirable state or it has not exhausted all specified recovery steps — which is a safety property. Unfortunately, this approach does not apply here, as, in our case, the protocol model is not meant to specify how an attack is mounted.

An alternative approach to a direct encoding is to show that any trace t can be combined with any trace t' that contains $p_1, \ldots, p_n$, c, and nothing else. This may hold for processes of a certain form. With such a result, protocol verifiers could again be used to show the existence of t'. For the present paper, we leave the verification of CC 1 as an open question.

Under this condition, and if we assume the set of traces to be prefix-closed, we obtain symbolic completeness.

Theorem 2. *If CS 1, CS 3, CS 4 and CC 1 hold, then* Π *is symbolically complete.*

Proof. Induction over the length of $pt|_{\Sigma_\cap}$.
Base case: $|pt|_{\Sigma_\cap}| = 0$: Holds trivially for $\sigma = \mathcal{I}$.
Inductive step: Let $pt\,|_{\Sigma_\cap} = e_1 ... e_k e_{k+1}$. From the IH and CS 3, we know that there exists a trace $t_k \approx e_1, \ldots, e_k \approx pt_k$. By definition of $\approx$, we know that $e_{k+1} \in Events$ and from $pt \in \mathcal{T}^\Pi$, we conclude that there is $a_{e_{k+1}} \in \mathcal{A}$ with e_{k+1} as a postcondition which was used to construct pt. By CS 1, $a_{e_{k+1}}$ = (pre, $\{e_{k+1}\}$). By CS 4 we know that $pre \subseteq \Sigma_\cap$.
The preconditions are met: $pre \subset set(t_k|_{\Sigma_\cap})$ because $pre \subset set(pt_k\,|_{\Sigma_\cap})$. Thus, we can apply CC 1 for $a = a_{e_{k+1}}$ and $t = t_k$ to obtain a trace $t' \approx t_k \circ t_r$ with $t_r|_{\Sigma_\cap} = e_{k+1}$. Hence $t' \approx t_k \circ e_{k+1} \approx e_1 ... e_k e_{k+1} \approx pt$. □

[2]According to their definition, 'no partial execution is irremediable since if some partial execution were irremediable, then it would be a "bad thing".'

To summarize: in conjunction, Theorem 1 and Theorem 2 ensure that the set of planning traces induced by Π and the set of protocol traces $\mathcal{T}$ are equal modulo $\Sigma_\cap$ if conditions CS 1 — CS 5 and CC 1 are met. This is necessary for risk estimation techniques that compute the expected loss of value or the probability of a breach.

CS 1 to CS 2 are satisfied w.l.o.g. for monotonic planning tasks and CS 3 is a standard assumption in protocol verification. CS 4 is a restriction we place on the composition of the security model and auxiliary models for the planning task. The remaining assumptions CS 5 and CC 1 are both trace properties, the former a safety property, the latter a liveness property. Given the lack of tool support, we will now focus on symbolic soundness, which ensures that that the planning model considers all possible attacks. If symbolic soundness holds, any quantitative result that is monotonic in $\mathcal{A}$ — e.g. the expected damage or the probability of reaching a critical asset — can be considered an upper bound, provided, of course, that model parameters such as the value of assets and probabilities of actions are correct.

V. Applications

The previous section results lay the foundation for using highly optimized planners for the analysis of large networks. We envision the following applications.

a) Protocol analysis for limited network attackers: Today's protocol verification focuses on protocols in isolation and against an attacker who can eavesdrop and modify all messages on the network. In terms of communication, this is the worst-case assumption for distributed services on the Internet. On the other hand, underlying services like the PKI or name resolution are almost always trusted and, more often than not, vastly simplified to the point of complete abstraction. For perspective, the Dolev-Yao model, which formalized these assumptions, is older than the first implementation of name resolution.

Planning models scale much better to large problem sizes (in terms of actions) than protocol verifiers, and are thus able to analyze the security of protocols in threat scenarios that are more complicated to describe. Incorporating more precise assumptions about the attacker could lead to more nuanced results, e.g. about protocol security in various topologies.

b) Cost-benefit guided protocol deployment in the Internet: Deployment assessment techniques are based on an infrastructure threat model and consider the deployment of a protocol as a 'countermeasure.' Using the recently proposed 'Stackelberg planning algorithm,' it is possible to obtain the set of all Pareto-optimal protocol deployments per node. This allows for an evaluation of the actual benefit of new proposals vis-à-vis the current infrastructure of the Internet. It makes it possible to compare proposals against each other that are incomparable on paper, e.g. is DNSSEC a better solution against JavaScript injection attacks than application-specific techniques like subresource integrity on the HTML level.

This technique has been applied to email [52] and the web [54], comparing solutions at the routing layer, resolution

layer and application layer. A weak point of this methodology was the lack of justification for their attacker model. Symbolic soundness and completeness can bridge this gap, as we will demonstrate for a subset of the email model [52]. As we argued in Section III, to justify the correctness of the Stackelberg planning problem, it is sufficient to show the symbolic soundness/correctness for the attacker planning problem, but for arbitrary initial states.

c) Corporate network analysis: Risk assessment techniques for local networks (e.g. mulVal [46]) focus on implementation-level flaws, e.g. buffer overruns, but often ignore the protocol level implications. An attacker that captures the company's certificate authority or authentication server can usually exploit this infrastructure's trust to obtain critical assets. Moreover, modern cloud-based services introduce new dependencies on external infrastructure. These aspects are rarely considered and could be improved by a rule-based representation of the involved protocol's flaws.

VI. Background: Email Case Study

We recall the email infrastructure attacker model by Speicher et al. [52] to justify its soundness in the next chapter. Using Stackelberg planning, they investigated how existing protocols can be used to secure users against large-scale eavesdropping by countries. While the impact of many techniques is different depending on the attacker and defender country (e.g. Russia and China are much more self-reliant than, e.g. Brazil), the enforcement of TLS and improved certificate validation have a significant impact throughout. In the following, we will focus on their threat model and infrastructure representation.

The email infrastructure is modeled as a *labeled property graph* [51], which is simply a graph with edge and node labels that describe service providers and their interdependencies.

Definition 4. *A labeled property graph is a directed multigraph and described as a quadruple $G = (V,E,\lambda,\mu)$ over an alphabet Σ. V is the set of nodes. $E \subset (V \times V)$ is a set of edges between nodes. The function $\lambda : V \cup E \to \Sigma$ maps a label from the alphabet Σ to nodes and edges. $\mu : (V \cup E) \times K \to S$ maps a string* value *$s \in S$ to a node/edge and a* key *$k \in K$.*

TABLE II: Node labels (top) and edge labels (bottom).

Labels	Description
IP	Node for IP address
Dom	Node for domain name.
AS	IANA number assigned to the AS.
Cntry	Country code
ORIG	AS where lhs node originates from
LOC	Country where lhs nodes is located
A	DNS record mapping Domain to Address
MX	DNS record mapping Domain to Domain
NS	DNS record for Name Servers
DNS	Resolving lhs requires resolving rhs
RES	lhs node uses resolver on rhs for resolution
RTE (AS_t)	AS-level route between ASes via AS_t

TABLE III: Corruption predicates

C (x)	Node $x \in$ Dom $\cup$ IP $\cup$ AS $\cup$ Cntry under attacker control
$\mathsf{I}^{\mathsf{DNS}}(d)$	Integrity of name resolution of $d \in$ Dom compromised
$\mathsf{I}^{\mathsf{R}}(d', e')$	Integrity of some route from $d' \in$ IP to $e' \in$ IP is compromised
$\mathsf{I}^{\mathsf{DNS}}(d, e)$	Integrity of name resolution of $e \in$ Dom from the perspective of $d \in$ Dom compromised
unconf(d, e)	email communication from some user of $d \in$ Provider to some user of $e \in$ Provider is considered unconfidential
nDNSSEC(d)	$d \in$ Dom does not support DNSSEC

Table II shows the node labels and edge labels used by Simeonovski et al. [51]. Figure 2 provides an example for the interaction between two mail providers. The green nodes represent IPv4 addresses and are labeled IP. They are associated to autonomous systems (orange, labeled AS) via the relation ORIG.

The blue nodes represent domain names and are labeled Dom. They are associated to IPs or other domains via the relations A, MX and NS which encode the resources records that were obtained by scanning. They designate the domain's IP address (A), its responsible mail server (MX) and its authoritative name server (NS), respectively.

The label DNS records the relationship between authoritative name servers and RES between mail servers and their resolvers. RTE(AS_b) is used to record routing dependencies. If AS_a is connected to AS_c and, somewhere along the way, a package might traverse AS_b, an attacker at AS_b could eavesdrop that communication. Domains, IPs and ASes are associated to countries via LOC edges.

A. Infrastructure attacker model

An attacker in this model can be a country or a group of countries that can corrupt all servers in their jurisdiction, as well as to observe, intercept and alter all messages routed through their jurisdiction.[3] The IA model tracks infrastructure compromise at different levels with corresponding predicates. For example, if the attacker has compromised the resolver of some domain, we would consider the *integrity* of all domain name resolutions of this domain compromised. However, the resolved domains themselves are not be compromised and may be safe to use for clients that use a different resolver. These predicates will be part of $\Sigma_\cap$ and thus have to coincide with corresponding events in the protocol model.

There are 16 rules (also called action schemas) that define how these predicates can be derived. They are parametric in the graph: for a given graph, they are compiled into a finite set of attacker actions $\mathcal{A}$ and predicates $\mathcal{P}$. Our focus is on the methodology; hence we will refer to Appendix A for the full set of attacker rules and only give a flavor of these rules with the following simple example.

[3] Attacks from large ISP can be modeled similarly [54].

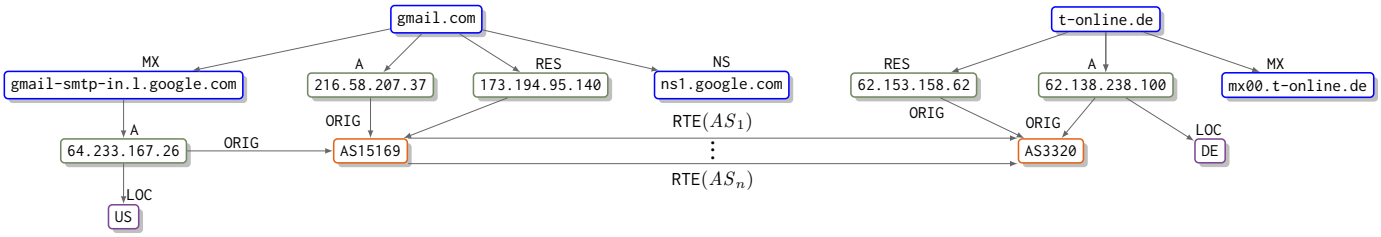

Fig. 2: Snippet of the property graph. (Taken from Figure 2 with permission of [52]).

Example 1.

$$\frac{d, e, r \in \text{Dom} \quad d \xrightarrow{\text{RES}} r \quad \mathsf{I}^{\mathsf{R}}(d, r) \quad \mathsf{nDNSSEC}(e)}{\mathsf{I}^{\mathsf{DNS}}(d, e)}$$

The intuition is as follows. If the attacker controls the route from a domain to the resolver that this domain uses, then we consider the integrity of any name resolution this domain attempts compromised. If the domain that is resolved uses DNSSEC, however, then the resolver can verify the integrity of this signature and this attack vector is not available. (A different rule deals with the case where the resolver itself is compromised.) The predicate nDNSSEC cannot be produced by the attacker, as it is a defender predicate.

In Speicher et al.'s model, all attacker rules that produce the predicate unconf are associated with a reward in terms of the number of users affected. The Stackelberg planning algorithm maximizes the sum of rewards. As $\mathsf{unconf} \in \Sigma_{\cap}$, the symbolic soundness result ensures that this is an upper bound.

B. Limitations

To simplify presentation, we concentrate on the core model, consisting of resolvers, DNS, DNSSEC and SMTP. We left out SMTP over TLS, DANE and IPsec for secure inter-AS communication. The protocol transformations we present in the next section would apply to the full model, as these protocols could be added without changing the structure of the processes. Thus the methodology would be the same, but the ProVerif processes would need to be extended.

The attacker model is not probabilistic, but relies on correct attacker rewards, which Speicher et al. [52] estimated from public sources. These are model parameters and need to be estimated.

VII. Background: ProVerif

In the following, we introduce ProVerif's dialect of the applied$-\pi$ calculus [10, 11]. Readers familiar with it can safely skip to the next section.

A. Syntax

We present the syntax of the calculus in Figure 3. Terms represent messages and data. Processes represent entities/programs. We use x, y, z to represent variables, a, b, c, n, k, s for free names and p, q for public names. We use FN and PN to refer to the set of free names and public names, respectively. Both are arbitrary, but infinite. The function symbol f represents a

$M, N ::=$	terms
v, x, y, z	variable
a, b, c, n, k, s	free name
p, q	public name
$f(M_1, ..., M_n)$	constructor application
$P, Q ::=$	processes
0	nil
$P \mid Q$	parallel
$!P$	replication
$\mathsf{in}(M, x).P$	input
$\mathsf{out}(M, N).P$	output
$(\nu a)P$	restriction
$\mathsf{if}\ M = N\ \mathsf{then}\ P$ $\mathsf{else}\ Q$	conditional
$\mathsf{let}\ x = g(M_1, ..., M_n)\ \mathsf{in}$ $P\ \mathsf{else}\ Q$	destructor application
$\mathsf{event}(M).P$	event

Fig. 3: Syntax of the process calculus

constructor whereas we use g to represent *destructors*. Both are abstract function symbols with some fixed arity.

Terms are defined over names, variables, and the applications of constructors. Destructors are used to manipulate terms in processes: $let\ x = g(M_1, ..., M_n)\ in\ P$ else Q binds x to the result of the destructor application of g on $M_1 ... M_n$ and continues with process P. If the application fails, however, we continue with process Q. A destructor g is defined by a finite set of reductions $def(g) := g(N_1, ..., N_n) \rightarrow N$ where the terms $N, N_1, ..., N_n$ are build without free names and $var(N) \subset var(N_1 \cup ... \cup N_N)$. A destructor fails, if no reduction applies.

Example 2. *Symmetric encryption is described by a 2-ary constructor* senc *and a 2-ary destructor with the following reduction:*

$$\mathsf{sdec}(\mathsf{senc}(x, y), y) \rightarrow x$$

We write $fn(P)$ (and $fv(P)$) for the sets of names (variables) that are free in P. A substitution $\delta = \{\{^{t_1}/_{x_1}\}, ..., \{^{t_n}/_{x_n}\}\}$ is a partial function, mapping variables to terms. The domain of δ is $\mathbb{D}(\delta) = \{x_1, ..., x_n\}$ and δ maps x_i to t_i. The application of g on the terms $M_1, ..., M_n$ is defined if and only if there exists some substitution δ and a reduction rule $g(N_1, ..., N_n) \rightarrow N$ such that for all $i \in \{1, ..., n\}$ it holds that $N_i = M_i\sigma$. In this case, let $x = g(M_1, ..., M_n)$ in P else Q

would bind x to $N\delta$ and continue to execute P.

Additionally, the process calculus provides the instruction $\mathsf{event}(F).P$ to emit some $F \in \Sigma_{Event}$ as an annotation of the process and continue to execute P. We define the set of these annotations as

$$Events := \{F(t_1, .., t_k) \mid t_i \text{ terms}, F \in \Sigma_{Event} \text{ with arity } k\}.$$

The remaining constructs depicted in Figure 3 are standard constructs included in the π-calculus. 0, or the nil process, indicates the end of the process and does nothing. $P|Q$ composes P and Q in parallel and $!P$ represents and unbounded number of copies of P in parallel composition. A *channel* can be any term M. The process $\mathsf{in}(M, x).P$ receives a message on channel M. It then continues to execute P with x being bound to the received message. $\mathsf{out}(M, N).P$ outputs a term N on channel M and executes P. $(\nu a)P$ depicts a restriction. It first creates a free name a and then executes P. A free name a is a secret and cannot be guessed, but it may be obtainable via computation/deduction of public messages. The conditional Q compares two terms M and N and executes P if they are equal and Q otherwise. For brevity, we will omit trailing 0 processes and empty else-branches.

B. Semantics

We define the semantics by first introducing the notions of *frame* and *deduction*. A frame $\nu\mathcal{E}.\delta$ represents a sequence of messages observed so far and the secrets generated by the protocol. The first is captured by a substitution δ, the latter by the set of used names $\mathcal{E}$.

Deduction describes the capabilities of an adversary to infer and compute new terms from already observed messages. We define the deduction relation $\nu\mathcal{E}.\delta \vdash t$ between a frame and a derivable term as the smallest relation s.t. the rules in Figure 5 hold. We further define f_{priv} as a subset of all constructor symbols where the DCON deduction rule cannot be used. We refer to f_{priv} as *private constructor symbols*.

The *operational semantics* are defined by a labeled transition relation between process configurations. This *configuration* is represented by a 3-tuple $(\mathcal{E}, \mathcal{P}, \delta)$. $\mathcal{P}$ is a multiset representation of processes being executed in parallel. $\mathcal{E}$ is the set of free names generated by the processes in $\mathcal{P}$. δ is a substitution modeling the messages observed by the environment.

The labeled transition relation of our calculus can be found in Figure 4. Each transition between two configurations is labeled with some $F \in Events \cup \{\varnothing\}$. For the ease of presentation, we omit empty sets and write $\rightarrow$ instead of $\xrightarrow{\varnothing}$. We define $\rightarrow^*$ to represent multiple application of transition rules labeled with the empty set. For the other $F \in Events$ we define $\overset{F}{\Rightarrow}$ as $\rightarrow^* \xrightarrow{F} \rightarrow^*$.

Definition 5 (Traces)**.** *Given a process P, we now define its traces:*

$$traces(P) = \Big\{(F_1, ...F_n) \mid (\varnothing, \{P\}, \varnothing) \overset{F_1}{\Longrightarrow} (\mathcal{E}_1, \mathcal{P}_1, \delta_1) \overset{F_2}{\Longrightarrow} \ldots \overset{F_n}{\Longrightarrow} (\mathcal{E}_n, \mathcal{P}_n, \delta_n)\Big\}$$

VIII. Case Study: Email

We now come back to Speicher et al.'s email case study (Section VI) to investigate the symbolic soundness of their model. Our focus will be on the methodology. We first present a translation from labeled property graphs into processes. Verifying this process for each property graph is impractical, both because of the size of the graph (protocol verifiers do not scale well with the model size) and because any change in the infrastructure would require a new analysis. Hence, we define two process transformations that allow for a sound mapping of all these processes to a single process, i.e. an over-approximation. We verify this process in ProVerif and can thus provide a symbolic soundness result for all process graphs at once.

A. Symbolic model

We define a function $\mathcal{F}$ from property graphs to processes in Figure 6. We use the following notation:

- For a finite set $S = \{a, .., z\}$, $\Big\|_{s \in S} P(s)$ denotes $P(a) \mid \ldots \mid P(z)$. Instead of $s \in S$, we sometimes use set-builder notation to directly define the components of each s.
- For a fixed labeled property graph G that is implicit in the context, we write V_x as a subset of all nodes in V with label x. We write $y \xrightarrow{L} z$ to represent an edge in G labelled with L connecting the two nodes x and y. To ease notation, we use d, e for nodes representing domain names, r for resolvers and n for name servers.
- We further assume that all nodes are public names, to avoid introducing a mapping.

Our process represents SMTP, DNS, DNSSEC, resolvers, and a simplified version of inter-AS communication. As we focus on the methodology, we do not elaborate on the subprocesses modeling these protocols, but on the top-level process that composes them. The processes $P_{\text{smtp-server}}$ and $P_{\text{smtp-client}}$ describe the client and server roles within the SMTP protocol. Each provider v defines several mail servers $d^{\text{c/s}}_{\mathsf{MX}}$ (or $e^{\text{c/s}}_{\mathsf{MX}}$) via the MX resource record. Each of those execute both client and server roles. They have one or many IP addresses $i^{\text{c/s}}_{\mathsf{MX}}$, which are located in autonomous systems $as^{\text{c/s}}_{\mathsf{MX}}$. Whereas the process $P_{\text{smtp-server}}$ only models the receiving part of the SMTP protocol, $P_{\text{smtp-client}}$ models DNS/DNSSEC requests as well as the client role of SMTP. To establish the connections between the different services, we use the IP addresses to model channels between them. These channels are built over private constructors. In contrast to the Dolev-Yao model, the attacker cannot eavesdrop or manipulate messages per default, but needs to obtain access to these channels by compromising either domain names, IP addresses, or ASes.

The processes P_{res}, P_{ns}, and P_{rns} describe the resolver and server role within the DNS protocol, which, depending on the server's configuration, include the DNSSEC extension. Process P_{res} models the resolver role by communicating with the DNS/DNSSEC infrastructure, on the one hand, and with the requesting role of the mail server. As with the previously

$$
\begin{aligned}
(\mathcal{E}, \mathcal{P} \cup^{\#} \{0\}, \delta) &\rightarrow (\mathcal{E}, \mathcal{P}, \delta) && \text{(NULL)}\\
(\mathcal{E}, \mathcal{P} \cup^{\#} \{P \mid Q\}, \delta) &\rightarrow (\mathcal{E}, \mathcal{P} \cup^{\#} \{P, Q\}, \delta) && \text{(PAR)}\\
(\mathcal{E}, \mathcal{P} \cup^{\#} \{!P\}, \delta) &\rightarrow (\mathcal{E}, \mathcal{P} \cup^{\#} \{P, !P\}, \delta) && \text{(REPL)}\\
(\mathcal{E}, \mathcal{P} \cup^{\#} \{\nu a; P\}, \delta) &\rightarrow (\mathcal{E} \cup \{b\}, \mathcal{P} \cup^{\#} \{P\{\{^{b}/_{a}\}\}\}, \delta) \text{ if b is free an not in } \mathcal{E} && \text{(NEW)}\\
(\mathcal{E}, \mathcal{P} \cup^{\#} \{\text{out}(t, M); P\}, \delta) &\rightarrow (\mathcal{E}, \mathcal{P} \cup^{\#} \{P\}, \delta \cup \{\{\{^{M}/_{x}\}\}\}) \text{ if x is fresh and } \nu\mathcal{E}.\delta \vdash t && \text{(OUT)}\\
(\mathcal{E}, \mathcal{P} \cup^{\#} \{\text{in}(t, x); P\}, \delta) &\rightarrow (\mathcal{E}, \mathcal{P} \cup^{\#} \{P\{\{^{M}/_{x}\}\}\}, \delta) \text{ if } \nu\mathcal{E}.\delta \vdash M \text{ and if } \nu\mathcal{E}.\delta \vdash t && \text{(IN)}\\
(\mathcal{E}, \mathcal{P} \cup^{\#} \{let\ x = M\ in\ P \text{ else } Q\}, \delta) &\rightarrow (\mathcal{E}, \mathcal{P} \cup^{\#} \{P\{\{^{M}/_{x}\}\}\}, \delta) \text{ if evaluation of M succeeds} && \text{(LETS)}\\
(\mathcal{E}, \mathcal{P} \cup^{\#} \{let\ x = M\ in\ P \text{ else } Q\}, \delta) &\rightarrow (\mathcal{E}, \mathcal{P} \cup^{\#} \{Q\}, \delta) \text{ if evaluation of M fails} && \text{(LETF)}\\
(\mathcal{E}, \mathcal{P} \cup^{\#} \{event(F); P\}, \delta) &\xrightarrow{F} (\mathcal{E}, \mathcal{P} \cup^{\#} \{P\}, \delta) && \text{(EVENT)}
\end{aligned}
$$

Fig. 4: Operational semantics.

Note that evaluation of some M succeeds, if for all destructor symbols in M, there is an applicable rewrite rule. If there is a destructor symbol in M which has no applicable rewrite rule, then evaluation fails.

$$
\frac{a \in FN \cup PN \;\; a \notin \overrightarrow{n}}{\nu\mathcal{E}.\delta \vdash a}\ (\text{DNAME}),\ \frac{x \in \mathbb{D}(\delta)}{\nu\mathcal{E}.\delta \vdash x\delta}\ (\text{DFRAME})
$$

$$
\frac{\nu\mathcal{E}.\delta \vdash t_1 \quad \ldots \quad \nu\mathcal{E}.\delta \vdash t_n \quad f \notin \mathsf{f}_{\text{priv}}}{\nu\mathcal{E}.\delta \vdash f(t_1, ..., t_n)}\ (\text{DCON})
$$

$$
\frac{\nu\mathcal{E}.\delta \vdash t_1 \ \ldots\ \nu\mathcal{E}.\delta \vdash t_n \quad \{d(t_1, ..., t_n) \rightarrow t\} \in def(g)}{\nu\mathcal{E}.\delta \vdash t}\ (\text{DDES})
$$

Fig. 5: Deduction rules.

mentioned process, the IPs are used to construct private channels via private constructors. The same holds for the name server role modeled by P_{ns}. An exception is the process P_{rns} modeling root name servers. We assume that root servers cannot be corrupted, since that would break the DNS/DNSSEC infrastructure as a whole. Therefore, the attacker is not able to corrupt the connection between the root server process and the process modeling the resolver role. Connections established by the processes P_{res} and P_{ns}, however, may be corrupted by corrupting their domain names, IP addresses, or ASes. For simplicity, we constrained our DNS model to two levels of name servers.

With this construction we represent the structure of the IA model. We instantiate all communication paths and relations in the IA model using the same labeled property graph G. Further, all featured protocols and functionalities of the IA model are represented by subprocesses in our model, as well as the notion of corruption.

In the follow up, we will modify the top-level structure, but leave the processes $P_{\text{smtp-client}}, P_{\text{smtp-server}}, P_{\text{res}}, P_{\text{ns}}$ and P_{rns} intact. They are detailed in the extended version [19, Appendix C].

B. Proof via sound process transformations

With $\mathcal{F}$, we can, in principle, verify the symbolic soundness of each planning task induced by some property graph G. The respective model $\mathcal{F}(G)$ can become very large: property graphs can have thousands to millions of nodes, whereas the majority of protocol models fits on a piece of paper. Protocol verifiers are not optimized for models of this size. Moreover, it is tedious to generate and verify a process whenever a new attacker country is considered or the property graph is modified. Last but not least, the analogy to computational soundness (Sec. II-3) suggest that symbolic soundness results (a) should encompass some set of protocols and (b) apply to any network composed of them, here described by the property graph. [4] Conceptually, we therefore desire a result that is independent of G.

To this end, we propose the following proof technique specific to the applied-π calculus. Let $\mathcal{F}(G)$ be a function from property graphs[5] to processes and assume that it can be expressed only using the applied-π calculus and the meta language operation $\Big\|_{s \in S} P(s)$. In the first step, we construct a process P such that, for all G, $traces(\mathcal{F}(G)) \subseteq traces(P)$. This implies that every trace property that holds for P also holds for $\mathcal{F}(G)$, independent of G. To this end, we apply to two transformations that over-approximates a process.

The first permits substituting several uniform parallel processes $\Big\|_{s \in S} P(s)$ by a single process under replication that obtains this input from the adversary. In the description of $\mathcal{F}(G)$, G can only occur within these S, hence the resulting process is now independent of G.

[4] Computational soundness results fix a set of cryptographic primitives, but hold for a class of protocols.

[5] We use property graphs for concreteness, the actual representation of the network is irrelevant, as long as it translates to planning models and processes in a uniform way.

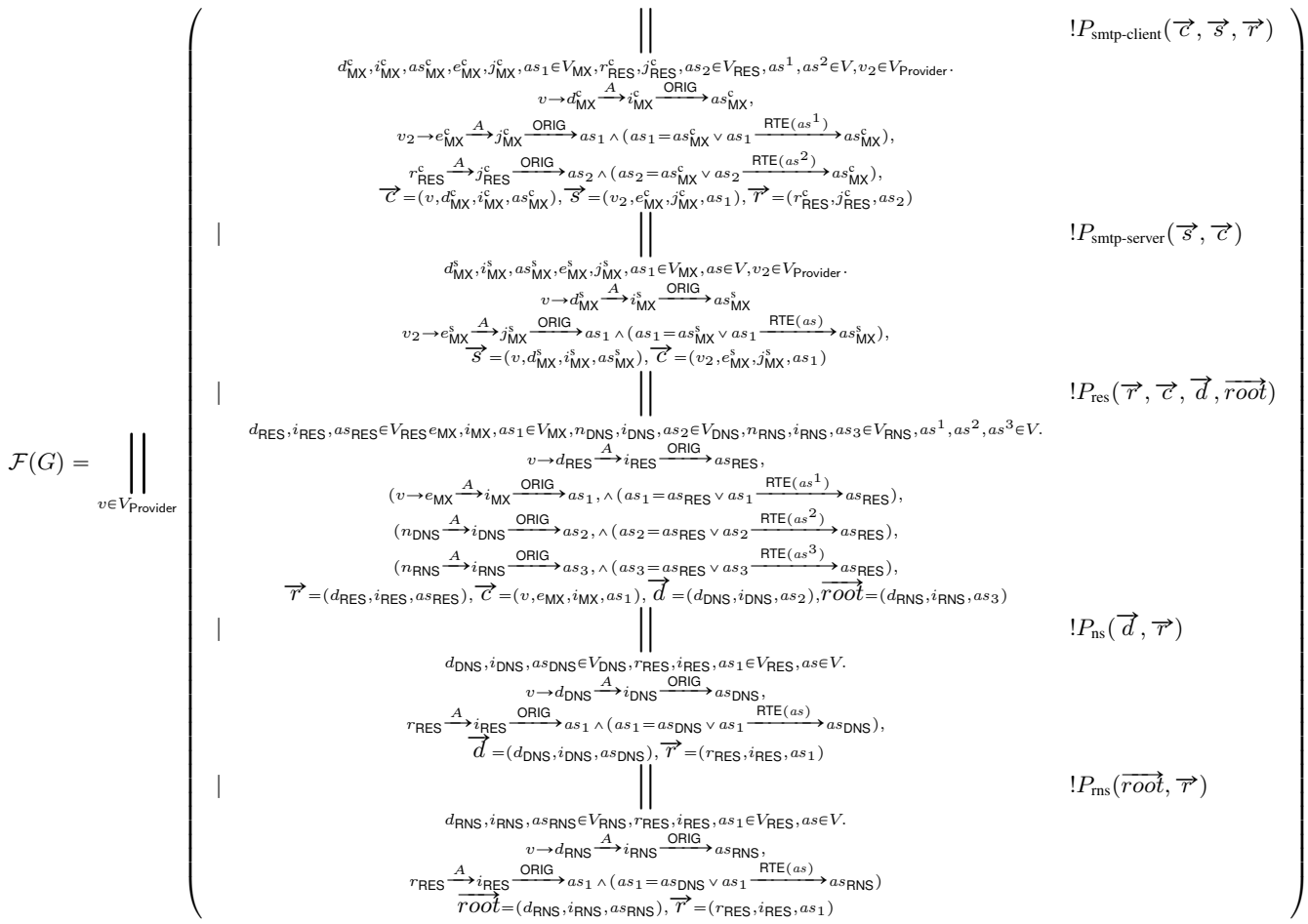

Fig. 6: Function $\mathcal{F}$ from property graphs to processes.

Lemma 1. *With*

$$traces(!\mathsf{in}(\overrightarrow{v}).P) \supseteq traces(\Big|\Big|_{\overrightarrow{p}} P\left\{{}^{\overrightarrow{p}}/_{\overrightarrow{v}}\right\})$$

we relate the replication of a process $in(\overrightarrow{v}).P$, where $\overrightarrow{v}$ can be supplied by the adversary (i.e., the frame), to a finite parallel execution of the same process P, where $\overrightarrow{v}$ gets substituted with public names supplied by G.

We may now automatically conduct the following verification steps with ProVerif (or any other verifier). ProVerif's abstraction in particular is sensitive to how deep in a process an input occurs. The second transformation thus permits pushing inputs further inside the process to aid verification.

Lemma 2. *For all processes Q that contain exactly one subprocess $\mathsf{in}(x).P$, let Q' be Q with P instead of this subprocess. Then: $traces(\mathsf{in}(x).Q') \subseteq traces(Q)$.*

Both lemmas are proven in Appendix B. In the second step, we ensure that these transformations have been correctly applied, i.e., that $traces(\mathcal{F}(G)) \subseteq traces(P)$ follows from Lemma 1 and 2. In the third step, we verify the three syntactic conditions on the planning task. Finally, we use ProVerif to show Condition CS 5. We partition $\mathcal{A}$ by postcondition. For this condition to hold, each element must have the following form.

$$\mathcal{A}_i = \{(pre_i^1, \{post_i\}), (pre_i^2, \{post_i\}), ..., (pre_i^k, \{post_i\})\}.$$

We translate each element into a correspondence query: for each trace, if $post_i$ occurs then pre_i^1, pre_i^2 or any other pre_i^j, $j \in \{1, \cdots, k\}$, occurs as well. As Condition CS 3 holds for any ProVerif process, we obtain symbolic soundness by Theorem 1.

C. ProVerif Model Introduction

Using the two lemmas above, we can transform *all* $F(G)$ into the process P, whose structure we present in Figure 7. The full model is in the extended version [19, Appendix C].

In this model, the adversary is also able to choose which processes communicate, and thus controls the underlying network topology. Given the process model based on the graph G in Figure 6 and our ProVerif model P, we show that the transformations have been applied correctly.

Theorem 3. $\forall G. traces(\mathcal{F}(G)) \subseteq traces(P)$.

```
!(in(c,prov:provider);
  !(in( dom_c:dom); in( ip_c:ip); in( AS_c:as);
    !(P_smtp-client(prov,dom_c,ip_c,AS_c)))
| !(in( dom_s:dom); in( ip_s:ip); in( AS_s:as);
    !(P_smtp-server(prov,dom_s,ip_s,AS_s)))
| !(in( dom_r:dom); in( ip_r:ip); in( AS_r:as);
    !(P_res(dom_r,ip_r,AS_r)))
| !(in( dom_d:dom); in( ip_d:ip); in( AS_d:as);
    !(P_ns(dom_d,ip_d,AS_d)))
| !(in( dom_rn:dom); in( ip_rn:ip); in( AS_rn:as);
    !(P_rns(dom_rn,ip_rn,AS_rn))))
```

Fig. 7: Simplified ProVerif model ($\mathsf{in}(m)$ short for $\mathsf{in}(c, m)$).

Lemmas 1 and 2 reduce the proof of this theorem to a structural argument (see Appendix B). The syntactic conditions on the planning task can be verified by inspecting it (Appendix A). Condition CS 1, discussed in Sec. IV-B, holds as there is no negated postconditions. Condition CS 2 holds, as all events in $\Sigma_\cap$ occur as a postcondition of some rule. Condition CS 4 holds, as all preconditions are in $\Sigma_\cap$.

D. Automated verification

It remains to show Condition CS 5, which we verify using ProVerif. The full set of queries is specified in the extended version [19, Appendix 1]. We grouped these queries according to whether the postcondition expresses a loss of integrity or a loss of confidentiality. We express the first property as a correspondence property and the second as a reachability property known as *weak secrecy*. For the first kind, we verify that any event matching some pattern e was preceded by an event matching a pattern e'. Any trace with an event matching e but not e' could be mapped to one where such integrity violation events are specifically marked; these would be the actual events in $\Sigma_\cap$. For the second kind, weak secrecy is expressed as usual. The attacker can demonstrate the ability to correctly input a secret message (in this case, the content of an email) in a subprocess. Upon success, the subprocess can be reduced to an event. We analyze the reachability of this event.

During the modeling process, we found two bugs in the IA model. First, in the IA model, $\mathsf{C}(ip)$ implies $\mathsf{C}(d)$ if d resolves to ip, but not vice versa. As $\mathsf{C}(ip)$ does not represent IP-level attacks, but a compromise of the service identified by ip, this ought to be the case. Without this rule, all rules that concern routing, name resolution or application compromise break down, as they identify the service with the domain it runs on. Luckily, this does not invalidate Speicher et al.'s result, as an inconsistency between the corruption of an IP and a domain can only come from (a) missing or inconsistent information in the property graph, e.g. domains d_1 and d_2 linked to different countries but resolving to the same IP, or (b) from an inconsistent initial network attacker state. We confirmed with the authors that neither condition was met.

The second bug concerns the DNSSEC protocol. DNSSEC requires resolver-side signature validation. This is not always the case for resolvers run by ISPs, but a realistic future scenario to investigate. By contrast, local resolvers (e.g. on clients or services like mail) rely on the ISP's validation during (recursive) resolving and will, at least for the near future, not validate signatures themselves. $r_{dns-route-res}$[52], however, assumes that this is the case, i.e. that DNSSEC is an effective countermeasure against domain poisoning attacks mounted between the local (recursive, usually non-validating) resolver and the ISP's (iterative, validating) resolver. Presumably, this is a bug, or at least an unrealistic assumption.

To solve the first problem, we added a rule turning $\mathsf{C}(d)$ into $\mathsf{C}(ip)$ and use $\mathsf{C}(ip)$ in all other rules, instead of $\mathsf{C}(d)$. To solve the second problem, we altered the rule by deleting the $\mathsf{nDNSSEC}$ predicate from the precondition. The changes are highlighted in Appendix A.

ProVerif proves all queries automatically and thus the last condition, CS 5. We used ProVerif version 2.02pl1. On an Intel i7-9750H CPU with 16 GB RAM, the analysis took 9.92s. This concludes our proof for the symbolic soundness.

E. Modeling Challenges

We take the opportunity to discuss some modeling challenges that we encountered and that are specific to our methodology.

The first is the modeling of the infrastructure attacker, who is less powerful than ProVerif's standard network attacker. It can only observe communication if the corresponding route has been compromised. Our first approach used *private channels* to model non-corrupted transfer of messages. We noticed ProVerif running into termination problems during the resolution. We minimized our model to three parties and found the issue to be ProVerif's internal representation of private channels as Horn clauses. This is because private channels are synchronous, as opposed to free (public) channels.[6] Routing in the Internet is actually asynchronous, so we model secret channels using 2-ary fact symbols req_packet, ans_packet and the following reduction:

$$\text{get\_req\_packet}(x, \text{req\_packet}(x, y)) = y.$$

(The reductions for ans_packet are analogous). All parties apply the function symbol with a shared key in the first parameter, to represent communication on that channel. The keys are built over names representing the IP addresses of the communicating parties, as well as a freshly chosen source port (sender) and the publicly known target port. To corrupt a key, the adversary claims the entity as part of its domain. Additionally, the adversary may choose some AS under its control and claims it to be part of the IP route between the communicating parties. With the corruption of this AS, the route is also seen as corrupted and the adversary can claim the key. This in-transit AS corruption model is very similar to the threat model described in the IA model.

The second challenge is how to structure the process such that information about corruption at the routing level is transmitted to processes that represent the resolution or application layer. As an example, imagine the adversary

[6] In addition, Babel, Cheval, and Kremer point out various communication semantics.

compromising an AS. All service providers affected would need to be informed that they can now output their keys. First attempts with private channels lead to non-termination. Instead, we restructured the process so that an AS compromise is a subprocess of the lower layer. Each entity needs to be compromised separately, but raises the same event $\mathsf{C}(as)$.

The third challenge is the size of the model. Using ProVerif's pretty printing, the process counts 360 lines, which is unusually large. The queries alone take about 47 lines. The most recent ProVerif release 2.02pl1 improved the verification time from 4 minutes (with 2.01) to about ten seconds. Hence we do not see a reason why the model could not be extended to cover Speicher et al. 's complete model at a reasonable level of abstraction. Nevertheless, a full-blown model of TLS could bring ProVerif to its limits. We suspect the model size is the reason why resolution takes unusually long — typically, ProVerif's analysis takes seconds or does not terminate at all. Disabling either the DNSSEC or DNS processes supports our suspicion that, the model size has a strong impact on the verification time, even though the models are relatively simple. A potential remedy is techniques for vertical and parallel composition (e.g. [21], [15], [28]), which could potentially be used to derive conditions for the composition of IA models.

IX. Conclusion

We introduced the first formal approach to justify vulnerability analysis and risk assessment techniques that operate on an Internet-wide scale. We provided a formal methodology to analyze a given model with off-the-shelf verifiers and demonstrated the applicability of our approach for symbolic soundness w.r.t. a real-world IA model. The protocol transformations and modeling tricks to represent infrastructure attackers in the Dolev-Yao model might be of independent interest for protocol analysis, e.g. for the analysis of p2p protocols.

We identify two main limitations: first, the verification of symbolic completeness requires either true[7] support for liveness in existing verifiers, or syntactical conditions that ensure that liveness can be concluded from reachability properties. We speculate that the reason for the lack of support is less in the technical challenges they pose, but the lack of a use case. Processes are expected to specify how a 'good state' can be reached.

The second limitation is the size of the model. We are confident that a holistic analysis of multiple protocols acting in parallel can be conducted for the whole of Speicher et al.'s model, but what if we want to include a full-grown model of different versions of TLS and IPsec? A deeper exploration of protocol composition in light of the infrastructure attacker and IP-like communication may yield a refined verification methodology and perhaps even composition results for IA models.

Acknowledgement

This research was partly supported by the ERC Synergy Grant "imPACT" (No. 610150).

[7] See discussion in Sec. IV-C.

References

[1] Martin Abadi and Cedric Fournet. "Mobile values, new names, and secure communication". In: *ACM Sigplan Notices* 36.3 (2001), pp. 104–115.

[2] Martín Abadi and Phillip Rogaway. "Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption)*". en. In: *J. Cryptology* 15.2 (2002), pp. 103–127. (Visited on 01/20/2020).

[3] Nawaf Alhebaishi, Lingyu Wang, and Anoop Singhal. "Threat Modeling for Cloud Infrastructures". In: *EAI Endorsed Transactions on Security and Safety* 5.17 (2018).

[4] Bowen Alpern and Fred B Schneider. "Defining liveness". In: *Information processing letters* 21.4 (1985), pp. 181–185.

[5] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. "Scalable, graph-based network vulnerability analysis". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM. 2002, pp. 217–224.

[6] Kushal Babel, Vincent Cheval, and Steve Kremer. "On the semantics of communications when verifying equivalence properties". In: *J. Comput. Secur.* 28.1 (2020), pp. 71–127.

[7] Michael Backes et al. "A Novel Approach for Reasoning about Liveness in Cryptographic Protocols and its Application to Fair Exchange". In: *Proceedings of the 2nd IEEE European Symposium on Security and Privacy (Euro S&P '17)*. IEEE Computer Society, 2017.

[8] Gilles Barthe et al. "Computer-Aided Security Proofs for the Working Cryptographer". In: *Advances in Cryptology*. Springer, 2011, pp. 71–90.

[9] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. "Verified models and reference implementations for the TLS 1.3 standard candidate". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 483–502.

[10] B. Blanchet. "An efficient cryptographic protocol verifier based on prolog rules". In: *Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001*. IEEE, 2001, pp. 82–96. (Visited on 01/20/2020).

[11] Bruno Blanchet et al. "ProVerif 2.00: automatic cryptographic protocol verifier, user manual and tutorial". In: *Version from* (2018), pp. 05–16.

[12] Mark S Boddy et al. "Course of Action Generation for Cyber Security Using Classical Planning." In: *ICAPS*. 2005, pp. 12–21.

[13] Tom Bylander. "The computational complexity of propositional STRIPS planning". In: *Artificial Intelligence* 69.1-2 (1994), pp. 165–204.

[14] Kaouthar Chetioui et al. "Formal Verification of Confidentiality in DNSSEC and E-DNSSEC Protocols Using Pi-Calculus and ProVerif". In: *Procedia Computer Science*. The 10th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2019) / The 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2019) / Affiliated Workshops 160 (1, 2019), pp. 752–757. (Visited on 07/08/2020).

[15] Vincent Cheval, Véronique Cortier, and Bogdan Warinschi. "Secure composition of PKIs with public key protocols". In: *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE. 2017, pp. 144–158.

[16] Ericson Clifton et al. "Fault tree analysis-a history". In: *Proceedings of the 17th International Systems Safety Conference*. 1999, pp. 1–9.

[17] Cas JF Cremers. "The Scyther Tool: Verification, falsification, and analysis of security protocols". In: *International conference on computer aided verification*. Springer. 2008, pp. 414–418.

[18] Cas Cremers et al. "A Comprehensive Symbolic Analysis of TLS 1.3". en. In: *Proceedings of the 2017 ACM SIGSAC*

Conference on Computer and Communications Security - CCS '17. ACM Press, 2017, pp. 1773–1788. (Visited on 02/06/2020).

[19] Alexander Dax and Robert Künnemann. *On the Soundness of Infrastructure Adversaries*. 2021.

[20] Danny Dolev and Andrew Yao. "On the security of public key protocols". In: *IEEE Transactions on information theory* 29.2 (1983), pp. 198–208.

[21] Santiago Escobar et al. "Sequential protocol composition in Maude-NPA". In: *European Symposium on Research in Computer Security*. Springer. 2010, pp. 303–318.

[22] Richard E. Fikes and Nils Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *AI* 2 (1971), pp. 189–208.

[23] Sylvain Frey et al. "It bends but would it break? topological analysis of bgp infrastructures in europe". In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2016, pp. 423–438.

[24] Nirnay Ghosh and S. K. Ghosh. "A Planner-Based Approach to Generate and Analyze Minimal Attack Graph". In: *Appl Intell* 36.2 (1, 2012), pp. 369–390. (Visited on 06/05/2020).

[25] Nirnay Ghosh and SK Ghosh. "An intelligent technique for generating minimal attack graph". In: *First Workshop on Intelligent Security (Security and Artificial Intelligence)(SecArt'09)*. 2009.

[26] Glenn Greenwald and Ewen MacAskill. "NSA Prism program taps into user data of Apple, Google and others". In: *The Guardian* 7.6 (2013), pp. 1–43.

[27] James A Hendler, Austin Tate, and Mark Drummond. "AI planning: Systems and techniques". In: *AI magazine* 11.2 (1990), pp. 61–61.

[28] Andreas V Hess, Sebastian A Mödersheim, and Achim D Brucker. "Stateful protocol composition". In: *European Symposium on Research in Computer Security*. Springer. 2018, pp. 427–446.

[29] Jörg Hoffmann. "Simulated Penetration Testing: From" Dijkstra" to" Turing Test++"". In: *Twenty-Fifth International Conference on Automated Planning and Scheduling*. 2015.

[30] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. "Practical attack graph generation for network defense". In: *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE. 2006, pp. 121–130.

[31] Florian Kammuller. "Verification of DNSsec Delegation Signatures". In: *2014 21st International Conference on Telecommunications (ICT)*. 2014 21st International Conference on Telecommunications (ICT). IEEE, 2014, pp. 298–392. (Visited on 07/08/2020).

[32] Kerem Kaynar and Fikret Sivrikaya. "Distributed attack graph generation". In: *IEEE Transactions on Dependable and Secure Computing* 13.5 (2015), pp. 519–532.

[33] Ekkart Kindler. "Safety and liveness properties: A survey". In: *Bulletin of the European Association for Theoretical Computer Science* 53.268-272 (1994), p. 30.

[34] Igor Kotenko and Mikhail Stepashkin. "Attack graph based evaluation of network security". In: *IFIP International Conference on Communications and Multimedia Security*. Springer. 2006, pp. 216–227.

[35] SSL Labs. *SSL Pulse*. URL: https://www.ssllabs.com/ssl-pulse/.

[36] Leslie Lamport. "Proving the correctness of multiprocess programs". In: *IEEE transactions on software engineering* 2 (1977), pp. 125–143.

[37] Changwei Liu, Anoop Singhal, and Duminda Wijesekera. "Using attack graphs in forensic examinations". In: *2012 Seventh International Conference on Availability, Reliability and Security*. IEEE. 2012, pp. 596–603.

[38] Heiko Mantel and Christian W Probst. "On the meaning and purpose of attack trees". In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE. 2019, pp. 184–18415.

[39] Bill Marczak et al. "An Analysis of China's "Great Cannon"". In: *5th USENIX Workshop on Free and Open Communications on the Internet (FOCI 15)*. USENIX Association, 2015.

[40] Bill Marczak et al. "China's great cannon". In: *Citizen Lab* 10 (2015).

[41] Simon Meier et al. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *Computer Aided Verification*. Springer Berlin Heidelberg, 2013, pp. 696–701. (Visited on 01/20/2020).

[42] A. Melnikov. "Updated Transport Layer Security (TLS) Server Identity Check Procedure for Email-Related Protocols". In: Request for Comments 7817 (2016).

[43] Merrill Morris and Christine Ogan. "The Internet as Mass Medium". In: *Journal of Computer-Mediated Communication* 1.4 (1996). JCMC141.

[44] MyEtherWallet. *A MESSAGE TO OUR COMMUNITY - a Response to the DNS HACK of April 24th 2018*. URL: https://medium.com/@myetherwallet/a-message-to-our-community-a - response - to - the - dns - hack - of - april - 24th - 2018 - 26cfe491d31c (visited on 06/04/2020).

[45] Jorge Lucangeli Obes, Carlos Sarraute, and Gerardo Richarte. "Attack planning in the real world". In: *arXiv preprint arXiv:1306.4044* (2013).

[46] Xinming Ou, Sudhakar Govindavajhala, and Andrew W Appel. "MulVAL: A Logic-based Network Security Analyzer." In: *USENIX security symposium*. Baltimore, MD. 2005, pp. 113–128.

[47] Cynthia Phillips and Laura Painton Swiler. "A graph-based system for network-vulnerability analysis". In: *Proceedings of the 1998 workshop on New security paradigms*. 1998, pp. 71–79.

[48] Eric Rescorla and Tim Dierks. "The transport layer security (TLS) protocol version 1.3". In: (2018).

[49] Bruce Schneier. *Schneier on Security - Attack Trees*. 1999. URL: https://www.schneier.com/academic/archives/1999/12/attack_trees.html.

[50] O. Sheyner et al. "Automated generation and analysis of attack graphs". In: *Proceedings 2002 IEEE Symposium on Security and Privacy*. IEEE Comput. Soc, 2002, pp. 273–284. (Visited on 02/06/2020).

[51] Milivoj Simeonovski et al. "Who Controls the Internet? Analyzing Global Threats using Property Graph Traversals". In: *Proc. of the 26rd International Conference on World Wide Web (WWW 2017)*. pub_id: 1147 Bibtex: SiPeRoBa_17:www URL date: None. 2017.

[52] Patrick Speicher et al. "Formally Reasoning about the Cost and Efficacy of Securing the Email Infrastructure". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 77–91.

[53] Patrick Speicher et al. "Stackelberg planning: Towards effective leader-follower state space search". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[54] Giorgio Di Tizio. "Pareto-Optimal Defensive Strategies Against JavaScript Injections". MA thesis. University of Trento, 2018.

[55] Yong-Jie Wang et al. "Study of network security evaluation based on attack graph model". In: *JOURNAL-CHINA INSTITUTE OF COMMUNICATIONS* 28.3 (2007), p. 29.

[56] Jianping Zeng et al. "Survey of Attack Graph Analysis Methods from the Perspective of Data and Knowledge Processing". In: *Security and Communication Networks* 2019 (2019).

APPENDIX

A. ProVerif queries

In this section, we will present the complete threat model described in [52]. Hence, the following will be freely, but completely cited from [52], except for the modifications marked in **bold orange**.

We will give each rule, followed by the intuition of what kind of attack it represents.

1) Initially Compromised Nodes:

Rule A.1 — $r_{init-loc}$ in [52] . All autonomous systems, IPs and domains associated to the attacking country are initially under control of the attacker.

$$\frac{x \in \text{AS} \cup \text{IP} \cup \text{Dom} \quad cn \in \text{Cntry} \quad x \xrightarrow{\text{LOC}} n \quad \mathsf{C}(cn)}{\mathsf{C}(x)}$$

Rule A.2 —$r_{init-as}$[52]. If an AS is under the control of the attacker, any IP which is part of the AS is also under control of the attacker.

$$\frac{i \in \text{IP} \; a \in \text{AS} \; i \xrightarrow{\text{ORIG}} a \; \mathsf{C}(a)}{\mathsf{C}(i)}$$

Rule A.3 —$r_{init-dom}$[52]. If an IP is under the control of the attacker, any domain that resolves to it (even if the attacker cannot interfere with the resolution) is also under the control of the attacker.

$$\frac{d \in \text{Dom} \; i \in \text{IP} \; d \xrightarrow{\text{A}} i \; \mathsf{C}(i)}{\mathsf{C}(d)}$$

Rule A.4 —$r_{init-ip}$ (this rule is new). If a domain is under the control of the attacker, any IP it resolves to (even if the attacker cannot interfere with the resolution) is also under the control of the attacker.

$$\frac{d \in \text{Dom} \; i \in \text{IP} \; d \xrightarrow{\text{A}} i \; \mathsf{C}(d)}{\mathsf{C}(i)}$$

2) Attacks via Routing:

Rule A.5 — $r_{injection}$[52] . If the attacker controls an AS which transfers packets from a domain m to some IP address belonging to n and this particular connection is not secured via the VPN mitigations, we assume that the integrity of the communication from $m \in \text{Dom}$ to $n \in \text{Dom}$ is compromised.

$$\frac{\sout{d,e \in \text{Dom}} \quad i,j \in \text{IP} \quad a,b,c \in \text{AS} \quad \mathsf{C}(b) \quad \mathsf{nVPN}(a,c) \quad \sout{d \xrightarrow{\text{A}} i} \quad \sout{e \xrightarrow{\text{A}} j} \quad i \xrightarrow{\text{ORIG}} a \quad j \xrightarrow{\text{ORIG}} c \quad a \xrightarrow{\text{RTE}(\langle\rangle b)} c}{\mathsf{I}^{\text{R}}(\mathbf{i, j})}$$

On the resolution and application level we are only concerned with communication between domains. Thus this rule covers all relevant routing attacks.

3) Integrity of domain/MX resolution:

Rule A.6 — r_{dns-ns}[52] . If the attacker controls any name server that could be queried during resolution, we consider the integrity of the domain name resolution compromised.

$$\frac{d, e \in \text{Dom} \quad d \xrightarrow{\text{DNS}} e \quad e \xrightarrow{\text{A}} i \quad \mathsf{C}(i)}{\mathsf{I}^{\text{DNS}}(d)}$$

Rule A.7 — $r_{dns-res}$[52] . If the attacker controls the resolver of a domain, we consider the integrity of any domain name resolution this domain attempts compromised. (Technically, r is an IP address, but we simplified this and the following rule for presentation.)

$$\frac{d, e \in \text{Dom} \quad i \in \text{IP} \quad d \xrightarrow{\text{RES}} i \quad \mathsf{C}(i)}{\mathsf{I}^{\text{DNS}}(d, e)}$$

Rule A.8 —$r_{dns-route-res}$[52] . If the attacker controls the route from a domain to the resolver this domain uses, we consider the integrity of any domain name resolution this domain attempts compromised ~~**, unless the integrity of the resolution is guaranteed by DNSSEC**~~.

$$\frac{d, e \in \text{Dom} \quad i \in \text{IP} \quad d \xrightarrow{\text{RES}} r \quad d \xrightarrow{\text{A}} i \quad \mathsf{I}^{\text{R}}(i, r) \quad \sout{\mathsf{nDNSSEC}(\mathbf{e})}}{\mathsf{I}^{\text{DNS}}(d, e)}$$

Rule A.9 — $r_{dns-route-ns}$[52] . If the attacker controls the route from a resolver to some authoritative name server potentially queried during resolution, we consider the integrity of the resolution for this domain name compromised, unless the integrity of the resolution is guaranteed by DNSSEC.

$$\frac{d,e,f \in \text{Dom} \quad r \in \text{IP} \quad d \xrightarrow{\text{RES}} r \quad e \xrightarrow{\text{DNS}} f \quad f \xrightarrow{\text{A}} i \quad \mathsf{I}^{\text{R}}(r, \mathbf{i}) \quad \mathsf{nDNSSEC}(e)}{\mathsf{I}^{\text{DNS}}(d, e)}$$

4) Confidentiality:

Rule A.10 — $r_{compromise}$[52] . If a mail server is already compromised, e.g., if it is hosted by an adversarial country, the attacker can compromise the confidentiality of the communication between two mail providers.

$$\frac{d,e \in \text{Provider} \quad d \xrightarrow{\text{MX}} d' \quad e \xrightarrow{\text{MX}} e' \quad d' \xrightarrow{\text{A}} d'' \quad e' \xrightarrow{\text{A}} e'' \quad \mathsf{C}(e'') \vee \mathsf{C}(d'')}{\mathsf{unconf}(d, e)}$$

Rule A.11 — $r_{fake-mx}$[52] . If the sender does not enforce strict host validation, e.g., by using optimistic STARTTLS, the attacker can compromise the confidentiality of the communication between two mail providers by changing a provider's MX record to point to a domain under her control.

$$\frac{d,e \in \text{Provider} \quad d \neq e \quad d \xrightarrow{\text{MX}} d' \quad \mathsf{nTLS}^{\text{snd}}(d) \quad \mathsf{I}^{\text{DNS}}(e) \vee \mathsf{I}^{\text{DNS}}(d', e)}{\mathsf{unconf}(d, e)}$$

Rule A.12 — $r_{fake-ip}$[52] . If the sender does not enforce strict host validation, the attacker can compromise the confidentiality of the communication between two mail providers

by pointing the domain of the MX to an IP of her choice.

$$\frac{d,e\in\mathsf{Provider} \quad d\neq e \quad d\xrightarrow{\mathsf{MX}}d' \quad e\xrightarrow{\mathsf{MX}}e' \quad \mathsf{I}^{\mathsf{DNS}}(e')\vee\mathsf{I}^{\mathsf{DNS}}(d',e') \quad \mathsf{nTLS}^{\mathsf{snd}}(d)}{\mathsf{unconf}(d,e)}$$

Rule A.13 — $r_{intercept}$[52] . If the sender does not enforce strict host validation, the attacker can compromise the confidentiality of the communication between two mail providers by intercepting packets on the route between their respective mail servers.

$$\frac{d,e\in\mathsf{Provider} \quad d\neq e \quad d\xrightarrow{\mathsf{MX}}d' \quad e\xrightarrow{\mathsf{MX}}e' \quad d'\xrightarrow{\mathsf{A}}d'' \quad e'\xrightarrow{\mathsf{A}}e'' \quad \mathsf{I}^{\mathsf{R}}(d'',e'') \quad \mathsf{nTLS}^{\mathsf{snd}}(d) \quad \mathsf{nDANE}^{\mathsf{rcv}}(e)}{\mathsf{unconf}(d,e)}$$

Rule A.14 — $r_{fake-mx-strict}$[52] . If the sender does not enforce certificate validation according to RFC 7817, e.g., by using optimistic STARTTLS or strict validation on the hostname only, the attacker can compromise the confidentiality of the communication between two mail providers by changing a provider's MX record to point to a domain under her control.

$$\frac{d,e\in\mathsf{Provider} \quad d\neq e \quad d\xrightarrow{\mathsf{MX}}d' \quad \mathsf{I}^{\mathsf{DNS}}(e)\vee\mathsf{I}^{\mathsf{DNS}}(d',e) \quad \mathsf{nRFC7817}(d)}{\mathsf{unconf}(d,e)}$$

B. Lemmas

Before proving Theorem 3, we present the proofs to the two process transformations from Section VIII-B .

Proof of Lemma 2. We can show that for every trace of the process $in(v).Q'$, the same trace can be produced by Q. In the first process, the adversary has to choose what term it binds to the free variable v in the beginning. Therefore, it needs to deduce T from the frame, s.t. $\nu\mathcal{E}.\delta \vdash t$. Comparing to Q, we can deduct the same term T by replacing $in(v).Q'$ with Q in the same configuration. Since no rule of our operational semantics deletes any information from the frame, we are able to deduce T at any point during the process execution of Q. This allows us to substitute v with T in both processes, leading to the same set of traces (since the rest of the processes is the same by construction.) □

Proof. To proof Lemma 1, we start by applying (repl) (see Figure 4) $\left|\overrightarrow{PN}\right|$ times to the process $!in(\overrightarrow{v}).P)$ and get a new process $Q = !in(\overrightarrow{v}).P) \mid \underbrace{in(\overrightarrow{v}).P) \mid ... \mid in(\overrightarrow{v}).P)}_{\left|\overrightarrow{PN}\right|}$. The variables $\overrightarrow{v}$ in the right process of Lemma 1 are substituted by public names provided by G. We apply the rule (in) also $\left|\overrightarrow{PN}\right|$ times on Q and the adversary can input the same public names as provided by G since all public names are deducible from the frame. With this transformation of Q we get exactly $!in(\overrightarrow{v}).P) \mid \Big|\Big|_{\overrightarrow{p}} P\{\frac{\overrightarrow{p}}{\overrightarrow{v}}\}$. Hence, we can conclude that

$$traces(!in(\overrightarrow{v}).P) = traces(!in(\overrightarrow{v}).P) \mid \Big|\Big|_{\overrightarrow{p}} P\{\frac{\overrightarrow{p}}{\overrightarrow{v}}\}) \supseteq traces(\Big|\Big|_{\overrightarrow{p}} P\{\frac{\overrightarrow{p}}{\overrightarrow{v}}\})$$

□

Proof of Theorem 3. First, we rearrange the $||$-quantification in $\mathcal{F}(G)$ (see Figure 6), so that $\overrightarrow{p}$ consists of all assignments to the meta-variables[8] x, y and z. (By definition, $||$ is associative and commutative.) For the reader's convenience, we index the applied-π variables with the meta-variable they replace.

$$traces(\mathcal{F}(G)) = traces\left(\Big|\Big|_{\overrightarrow{p}} P_{\text{proto}}\left\{\frac{\overrightarrow{p}}{\overrightarrow{v}}\right\}\right),$$

where $P_{\text{proto}} = !P_{\text{smtp-client}}(\overrightarrow{x}^{\text{c}}, \overrightarrow{x}^{\text{s}}, \overrightarrow{x}^{\mathsf{RES}}) \mid !P_{\text{smtp-server}}(\overrightarrow{y}^{\text{s}}, \overrightarrow{y}^{\text{c}}) \mid !P_{\text{res}}(\overrightarrow{z}^{\mathsf{RES}}, \overrightarrow{z}^{\text{c}}, \overrightarrow{z}^{\mathsf{DNS}}, \overrightarrow{z}^{\mathsf{RNS}}) \mid !P_{\text{ns}}(\overrightarrow{u}^{\mathsf{DNS}}, \overrightarrow{u}^{\mathsf{RES}}) \mid !P_{\text{rns}}(\overrightarrow{w}^{\mathsf{RNS}}, \overrightarrow{w}^{\mathsf{RES}}))$ (compare with Figure 6). Therefore, by applying Lemma 1 we get:

$$\subseteq traces\,(!\mathsf{in}(\overrightarrow{v}).P_{\text{proto}})$$

Note that all variables are uniquely named. We can hence exhaustively apply Lemma 2 to P to push all variables in $\overrightarrow{v}$ to the inside far enough that the resulting process P'_{proto} equals P (compare with full model in the extended version [19, Appendix C].). Verifying the syntactical equivalence, we obtain:

$$\subseteq traces(P).$$

□

[8]The $||$-notation is a syntactic shortcut on the mathematical level, hence the variables it binds are mathematical variables, not ProVerif variables. They stand for nodes in the graph, which we assumed to be public names.