Towards a Secure and Verifiable Future Internet (Full Presentation)

Limin Jia[†] Chen Chen[*] Sangeetha A. Jyothi[*] Wenchao Zhou[*] Suyog Mapara[*] Boon Thau Loo[*]

[*] *University of Pennsylvania* [†] *Carnegie Mellon University*

1. INTRODUCTION

In recent years, there have been strong interests in the networking community in designing new Internet architectures. One of the driving forces behind these "clean-slate" designs is the need to address pressing security concerns of the Internet. As a consequence, recent architectures, such as SCION [8] and ICING [5], provide radically new architectures that claim to provide stronger security guarantees. One of the limitations of current proposals is that security claims of these new designs lack formal security proofs – these protocols are evaluated primarily via experimental evaluations and argued via informal reasoning.

In this paper, we outline our research agenda on programming language support for implementing secure Internet protocols, and verifying the security properties of these implementations. Central to our work is the use of *Secure Network Datalog* (*SeNDLog*) [9], a declarative networking [4] language with cryptographic primitives. SeNDLog extends the *Network Datalog* (*NDLog*) declarative networking language with user-defined cryptographic functions.

Specifically, our project aims to achieve the following goals. First, we plan to demonstrate that most existing secure Internet routing architectures can be easily expressed in SeNDLog. Second, to facilitate formal proofs of security, we are developing a set of sound reasoning principles over SeNDLog. Using these reasoning principles, we would be able to extract proof obligations in the form of first-order logic formulas given any SeNDLog program and the security properties in question.

2. CASE STUDY: INTERNET ROUTING

BGP assumes a network model in which routers are grouped into various Autonomous Systems (*AS*) administrated by Internet Server Provider (*ISP*). Individual ASes exchange route advertisements with neighboring ASes using the *path-vector* protocol. Since these route advertisements are not authenticated, ASes can lie and advertise non-existent routes, or claim to own destination addresses that they do not. These faults may be a consequence of harmless misconfigurations, or malicious activities, e.g. traffic hijacking or violations of business agreements.

To address these problems, there have been a variety of proposed mechanisms [2] that aim to improve the trustworthiness of the route announcements. For example, Secure BGP [3] and Secure-Origin BGP [7] use cryptographic functions to sign routing information to prevent malicious routers from altering the routing information.

3. SECURE PROTOCOLS IN SeNDLog

To demonstrate the key language features of SeNDLog, we present a SeNDLog program that computes the reachability of all pairs of nodes.

```
At S:
s1 reachable(S,D) :- neighbor(S,D).
s2 S says reachable(Z,D)@Z :- neighbor(S,Z),
      W says reachable(S,D).
```

The rules `s1` and `s2` are located in the context of principal `S`, and specify a distributed transitive closure computation, where rule `s1` computes all pairs of nodes reachable within a single hop, and rule `s2` expresses that "if `Z` is a neighbor of `S`, and `W` claims that `S` can reach `D`, then `S` will inform `Z` about its reachability to `D`". The `says` primitive in the body of rule `s2` specifies that the authenticity of the received `reachable` tuple should be checked, ensuring that it originated from `W`. On the other hand, the `says` primitive in the head specifies that node `S` should digitally sign that tuple before sending it to `Z`. In the implementation, the `says` primitive corresponds to operations of generating and checking digital signatures. The location specifier `@Z` in rule `s2` indicates that the evaluation result should be communicated to node `Z`.

In the above program, each neighbor authenticates routes received from immediate neighbors. The actual S-BGP and so-BGP can be encoded by extending this program to enable the authentication (via signatures) of all subpaths within an advertised path. Based on our experience in S-BGP and so-BGP, we are in the process of making the following extensions to SeNDLog:

User-defined cryptographic functions. Depending on the security requirements, programmers can choose the cryptographic functions that provide the right tradeoff between security guarantees and computational overhead.

Send and receive actions. The second extension is explicit language constructs for send and receive actions. For sending, we can use the existing location operator @. When the location of the head differs from the location of the body, the rule head is interpreted as a send action. Similarly, to truthfully model the adversary abilities, we further distinguish between tuples that can be received from other nodes, and tuples local to the node. We use $b \leftarrow u$ to denote that tuple b is received from the network. We allow programmers to declare whether the tuple is private, or can be received from the network. Nodes will not accept tuples sent over the network if they are declared as private.

Rule ordering. In SeNDLog, when two rules can be fired, one will be picked non-deterministically. However, rule ordering is essential for precisely capturing the transitions of protocols steps. We are working on resolving rule ordering.

4. VERIFYING SeNDLog PROTOCOLS

Unlike traditional cryptographic protocols (e.g. authentication protocols) that typically involve a small finite number of nodes, network protocols involve an arbitrarily large number of participants. Each router in the protocol may contain local states, such as neighborhood and routing tables. The correctness of the protocol as a whole depends on these local states being maintained correctly. This suggests that one should adopt inductive reasoning of invariants of the entire execution history of the protocol, which is not commonly seen in cryptographic protocols.

We outline our initial thoughts on verification of SeNDLog programs. Concurrently, we are encoding S-BGP and so-BGP protocols in π-calculus, and verifying shallow properties within the ProVerif [1] tool. Interestingly, ProVerif's translation of π-calculus specifications to horn clauses bears resemblances to our SeNDLog programs, suggesting the potential of unifying the two approaches in the near future.

4.1 Formal Semantic Models

Routing protocols written in SeNDLog specify transition systems, where a system state is typically composed of the local state of each node, and transitions between states are triggered by events. We can specify the properties of the behaviors of these transition systems in terms of the properties of the execution traces generated by these systems.

We write *prog* to denote a SeNDLog program, and Σ to denote system state. Each state Σ is composed of a set of events, written E, waiting to be processed, and the local state s of each node: $\Sigma = (E, s_{u1}, \cdots s_{uk})$. We index each local state by the name of the node. For instance s_u is the state for node u. Both E and s_{uk} are a set of predicates.

A one-step transition of a program *prog* is represented as $E, s \xrightarrow{\textit{prog}} E', s'$, where the program *prog* is triggered by events E, transits from state s to s', and generates events E'. Routing protocols often involve programs running on several nodes concurrently. We write $\mathcal{P}$ to denote the set of programs in the system ($\mathcal{P} = \{\textit{prog}_{u1}, \cdots, \textit{prog}_{un}\}$). We write $\Sigma \xrightarrow{\mathcal{P}} \Sigma'$ to denote the one step transition of the system. The system takes a step if a program $\textit{prog}_{uj}$ takes a step. Given the specifications of the routing protocols, the behavior of one run of the protocols can be captured in terms of the trace generated: $\Sigma_0 \xrightarrow{\mathcal{P}} \Sigma_1 \cdots \xrightarrow{\mathcal{P}} \Sigma_n \cdots$. Note that some of the programs in $\mathcal{P}$ may be malicious.

One challenge is to correctly interpret SeNDLog programs in terms of execution traces. The most natural semantics for SeNDLog will be the fixed-point semantics: the set of facts that a SeNDLog program can derive. However, the facts contain not only logical facts, but also events such as sending and receiving tuples. The simple fixed point view is too coarse-grained. To this end, we plan to build on prior work on developing low-level semantics for NDLog [6], and extend it to support SeNDLog features.

4.2 Identifying and Specifying Properties

Unlike cryptographic protocols, which have been well studied, there is little formal account of the security properties of the Internet routing architectures. We hope to identify these properties, and state them formally in logic so that they can be used as standard properties to check on new designs.

Linear Temporal Logic (LTL) can concisely specify properties of execution traces of transition systems. One property of a secure routing protocol is that for any route announced by an honest router (routers that run the unaltered protocol), the first node originates this path should be the owner of the announced prefix. The logical formula is presented below. $\Box\varphi$ means that φ has to be true in all states of the trace.

$$\Box(\forall p.\forall v.\mathsf{send}\,(v,p) \wedge \mathsf{honest}\,(v) \wedge (p = \mathsf{concat}\,(p', [u,d])) \supset \mathsf{owns}\,(u,d)$$

It is well know that LTL can be translated to first-order logic by augmenting predicates with an additional time argument indicating when the predicate is true. In our reasoning, we actually use first-order logic directly.

4.3 Reasoning about SeNDLog Programs

We do not know the code that malicious routers run. Therefore, given a routing protocol written in SeNDLog, we need to reason about its properties in the presence of unknown adversaries. Fortunately, the capabilities of the adversary is known, and these capabilities can be specified as axioms.

We plan to develop a set of reasoning principals for SeNDLog programs. One promising idea is to directly generate first-order logic formulas describing the behavior of the program. For instance, given a rule of the form $\forall\vec{x}.h(\vec{x})$:- $b_1(\vec{y_1}), \cdots b_n(\vec{y_n})$. its behavior is captured by the following formula:

$$\forall\vec{x}\forall t, h(\vec{x},t) \supset (\exists\vec{y_1}\exists t'_1.b_1(\vec{y_1},t'_1) \wedge t'_1 < t) \vee \cdots \vee (\exists\vec{y_n}\exists t'_n.b_n(\vec{y_n},t'_n) \wedge t'_n < t)\ .$$

(We augment each tuple with an additional time field.)

Given the formulas that encode the program's behavior, the axioms about adversary capabilities, and the security properties of interest, we can then try to use a first-order logic theorem prover to discharge the proof obligation. One drawback of such an approach is that we are not specifying that the derivation is triggered by the freshest values of the tuples. That is a property of the derivation, which is hard to express in pure logical form. We are currently looking into encoding the freshness information as axioms as well.

Acknowledgements. This work is funded in part by NSF grants IIS-0812270 and CNS-1117052.

5. REFERENCES

[1] Bruno Blanchet and Ben Smyth. ProVerif 1.86: Automatic cryptographic protocol verifier, user manual and tutorial. http://www.proverif.ens.fr/manual.pdf.

[2] Matthew Caesar and Jennifer Rexford. BGP Routing Policies in ISP Networks. In *IEEE Network Magazine, special issue on Interdomain Routing*. 2005.

[3] Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure border gateway protocol (S-BGP). In *IEEE Journal on Selected Areas in Communications*, 18:103–116, 2000.

[4] Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking. In *Communications of the ACM*, 2009.

[5] Jad Naous, Michael Walfish, Antonio Nicolosi, David Mazieres, Michael Miller, and Arun Seehra. Verifying and enforcing network paths with ICING. In *CoNEXT*, 2011.

[6] Vivek Nigam, Limin Jia, Boon Thau Loo, and Andre Scedrov. Maintaining distributed logic programs incrementally. In *PPDP*, 2011.

[7] Russ White. Securing BGP through secure origin BGP (so-BGP). *The Internet Protocol Journal*, 6(3):15–22, 2003.

[8] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G. Andersen. Scion: Scalability, control, and isolation on next-generation networks. In *Oakland S&P*, 2011.

[9] Wenchao Zhou, Yun Mao, Boon Thau Loo, and Martín Abadi. Unified Declarative Platform for Secure Networked Information Systems. In *ICDE*, 2009.