Scion Design & Development Team

Verification Team

Information Security
David Basin
Tobias Klenze
Ralf Sasse
Christoph Sprenger
Thilo Weghorn

Programming Methodology
Marco Eilers
Peter Müller

Network Security
Samuel Hitz
Adrian Perrig
Motivation and Context

Routing problems with the status quo (inter-AS routing)
The Internet is a network of **Autonomous Systems (ASes)**.

Each AS is itself a network of **routers** run by an institution (e.g., Telco, ISP, company, or university).

There are 50,000+ ASes in the world.
Autonomous systems and routers

1. finds paths between ASes (and within ASes)
2. forwards data along paths

• Multiple paths between ASes: 2,1,4 and 2,3,4

• Computed in background by *Border Gateway Protocol* (BGP) and just one will be selected and used to configure routers
Border Gateway Protocol

• ASes announce paths to destination address ranges.
  - One path per destination used to configure routers.
  - Data flows back in the opposite direction.

• Policies
  - Decide on what is accepted, rejected, or propagated.
  - Any AS can announce any address range it wants!

My network, my rules!

Destination ip: some IP address range, e.g., 208.65.153.0/24

It’s all based on T.R.U.S.T.
Who controls the Internet?

- Control over paths is completely distributed
  - Border Gateway Protocol (BGP): all nodes flood path announcements
- No inbound traffic control
Who controls Internet paths?

Traceroute Path 4: from Chicago, IL to Tehran, Iran
Three concrete examples

Pakistan DoS against Youtube (2 hours, 2008)

Strange snafu hijacks UK nuke maker’s traffic, routes it through Ukraine
Lockheed, banks, and helicopter designer also affected by border gateway mishap.

Redirected traffic to UK Atomic Weapons Establishment

Ukraine ISP hijacks UK routes including UK Atomic Weapons

Fribourg’s government address space stolen for 3 days by SPAMers
Scion

Routing as it should be
SCION Project
Secure Future Internet Architecture

- Design & Implementation, 75+ man years
- Design of routing / forwarding protocols, support ecosystem, and numerous extensions
- Clean slate, yet compatible with existing Internet
- Not just a research prototype, growing deployment: 26 ASes on 3 continents

- See [www.scion-architecture.net](http://www.scion-architecture.net) and related publications
SCION Overview

- **Isolation Domains (ISD)**
- **Control Plane**: routing
  - Path exploration
  - Path registration
  - Path resolution
- **Data Plane**: packet forwarding
SCION Isolation Domain (ISD)

1. Agreement:
   Each region agrees on a common trust root.

2. Failure Isolation:
   No ISD can influence another ISD’s control plane.
Routing Phases:
(1) Path Exploration
(2) Path Registration
(3) Path Resolution

• Path Construction Beacons (PCB) are Sequence of signed Hop Fields
• Hop Fields (HF) carry the routing information for one AS
Routing Phases:
(1) Path Exploration
(2) Path Registration
(3) Path Resolution

- Path Construction Beacons (PCB) are Sequence of signed Hop Fields
- Hop Fields (HF) carry the routing information for one AS

Beaconing

<table>
<thead>
<tr>
<th>PCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core: Out: 4</td>
</tr>
<tr>
<td>AS E: In: 1, Out: 4</td>
</tr>
<tr>
<td>AS F: In: 1, Out: 3</td>
</tr>
<tr>
<td>AS B: In: 1</td>
</tr>
</tbody>
</table>
SCION Routing (Control Plane)

Routing Phases:
(1) Path Exploration
(2) Path Registration
(3) Path Resolution

Routing Phases:
(1) Path Exploration
(2) Path Registration
(3) Path Resolution
SCION Routing (Control Plane)

Routing Phases:
(1) Path Exploration
(2) Path Registration
(3) Path Resolution
Packet header

Forwarding along:
- Up-Segment
- Core-Segment
- Down-Segment

Segments are sequences of Hop Field (HFs).

Hop Field contain routing information of one AS.

SCION Forwarding (Data Plane)
Verification

High-level, omitting formal details
Can We Verify Scion?

- Control and data plane guarantees
- Functional correctness of actual code
  - Suitable for high-assurance business cases
  - Ensures that routers are backdoor-free
- Scion routers are simple and stateless
  - This is the key to their (feasible) verification
  - Not possible for current Internet with highly complex routers and giant code bases of millions of lines
Correctness and Security
SCION Approach

Verification of protocols models at the network level

- **Assuming** that each SCION component behaves as specified
- **Technique**: stepwise refinement, preserves invariants, using Isabelle/HOL.

Verification of the components at the code level

- **Guaranteeing** that each SCION component behaves as specified.
- **Technique**: Hoare-style pre-/post-condition reasoning, Viper with Python front-end.
Concrete Attacker Model

We use a **localized, colluding** Dolev-Yao attacker model

Attacker controls the entire network

Attacker controls a subset of ASes
SCION Protocol Security Properties

Control plane properties

• **Beacon validity:** Sequence of ASes in a beacon corresponds to a path in the network (modulo wormholes).

Data plane properties

• **Path authorization:** Packets only forwarded along previously authorized paths.

• **Weak detectability:** An active attacker cannot hide his presence on the path.

Our initial focus is on data plane / router code verification.
System & Environment

Environment

System

Attacker
Network
End hosts
OS & Libraries
Border Router
SCION Router Verification Overview

Model

Environment Model
attacker, network

Router Model

Reality

Real Environment

Router Code

Protocol Security Properties

Code Security Properties

satisfies

refined by

unproven

justified

proven

Verified SCION
SCION Router Verification Overview

Model

Environment Model
attacker, network

Router Model

Reality

Real Environment

Router Code

Protocol
Security Properties

Code
Security Properties

satisfies
refined by
unproven
justified
proven
The Path is the Packet

- A path is a sequence of **Hop Fields (HF)**.
- Each Hop Field contains routing information for one AS.
- Path is separated into Past and Future parts that indicate the packet’s position in network.
Refinement Overview

Communication channels  Hop Field format  Attacker

Idea: strengthen attacker while increasing protection of paths.

Message set  Neighbor ASes  MAC  Fields protected by MAC
Simplified Scenario (Initially)
Packet traversal along a single up-segment

- A set of **authorized-paths** from path server is given as parameter

- Simplified setting
  - Ignore core- and down-segments
  - No peering or core links
  - Single ISD
  - No changes in link status (up/down)

**Verification is still challenging enough!**
### Data Plane Model 0

**Example of one Packet along a simple Path**

<table>
<thead>
<tr>
<th>Past Path</th>
<th>Future Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

**Recv0**

![Diagram showing A, B, and C]

**Up0**

![Diagram showing A, B, and C]

**Send0**

![Diagram showing A, B, and C]
Data Plane Model 0

Problem: Past Path is unreliable
• Add new component, **real path**, to message

• History variable recording **actually path traversed** so far (Not part of actual implementation)
Assumption (control plane) Assume a set of authorized-paths resulting from beaconing process.

- **Path authorization**: Packets are forwarded only along previously authorized paths.

- **Weak detectability** An attacker 🧟‍♂️ cannot hide his presence on the path; follows from the following suffix property:
Data Plane Model 1

Real Path → Past Path → Future Path

Hop Field format is refined:

Model 0

A

Model 1

(,<,<,A,<,>)

Added: references to previous and next AS
Data Plane Model 2: "Chaining" of MACs

Hop Field format is further refined by adding a MAC

- MAC at A is produced with a key(A) known only to A
- MAC includes data and MAC of subsequent Hop Field (needed for verification)

Simplified representation:

\[(\bot, A, \square, \bullet) (\nabla, B, \square, \bullet) (\nabla, C, \square, \bullet) (\nabla, D, \bot, \bullet)\]
Up-Event in Model 2

Guard

\[ \text{In select in} \]

Condition

\[ \text{Check} \]

\[ \begin{array}{c}
\text{between } (\bullet_1, A_1, \bullet_1, 1) = (\bullet_2, A_2, \bullet_2, 2) \\
\land \bullet_1 = \text{valid MAC using key}(A_1) \\
\land \bullet_2 = \text{valid MAC using key}(A_2) \\
\land \bullet_1 = A_2 \land \bullet_2 = A_1
\end{array} \]

Action

\[ \text{Out put } \text{message} \text{' in where} \]

\[ \begin{array}{c}
\text{message} = (\bullet_1, A_1, \bullet_1, 1) \\
\text{message} = (\bullet_2, A_2, \bullet_2, 2)
\end{array} \]
Refining Model 2

Model 2

Global Message Set

Inter-AS Message Sets

Model 3
Up-Event in Model 3

Guard

\[ \text{In select from } \]

Check

\[ \text{Guard} = \ldots (\text{key}(A_1)) \ldots \]

\[ \land \quad \bullet_1 = \text{valid MAC using key}(A_1) \]

\[ \land \quad \bullet_1 = \text{key} \quad \land \quad \bullet_1 = \text{key} \]

Action

\[ \text{Out put } \text{in } \]

\[ \text{Action} = \ldots (\text{key}(A_1)) \ldots \]
SCION Router Verification Overview

Model

Environment Model
attacker, network

Router Model

Reality

Real Environment

Router Code

Protocol Security Properties

Code Security Properties

Verified SCION

satisfies
refined by
unproven justified proven
Router Model vs. Code

Model

Environment Model
attacker, network

Router Model

Reality

Real Environment

Router Code

Guard

In (,,)
Check (,,)
Action (,,)

Out (,,)

def router():
    while (pkt.next()):
        pkt.process()
    ...

42
**Code-Level Verification**

- **Main goal:** prove **functional correctness**.
  - Code refines the protocol.

- **Other desirable properties only on code level:**
  - **Safety:** Code does not raise runtime exceptions or have data races.
  - **Secure information flow:** Code does not leak any information about crypto keys.
  - **Liveness and deadlock freedom**

- **Focus on the SCION code base.**
  - Used libraries are given specifications, **assumed** to be correct.
  - Runtime, OS, ..., are **assumed** to be correct.
Program Verification

- **Formal specification** for each method
  - Pre- and postcondition, loop invariants

- **Formal proof** that implementation satisfies specification.
  - Assuming **precondition** holds at the beginning, prove that **postcondition** holds after return (partial correctness).
  - For all possible inputs, schedules, callers, ...
  - Additional proof obligations for special properties, like progress

```python
def sqrt(n):
    ...
    return result
```
Code-based Verification

- Scion in Python 3
  - ~11k LOC

- Substantial subset of Python
  - Most standard OOP features
  - e.g. inheritance, exceptions, concurrency

- Focus on router first

- Use Viper Toolchain with Python front end
Linking it all up via Input-Output Specifications
(Code can be viewed as a transition system)

Guard

In  \[ \boxed{\ldots} \quad ( \ldots ) \quad ( \ldots ) \]

Check

\[ \implies \ldots \]

\[ \boxed{\ldots} \quad ( \ldots ) \quad ( \ldots ) \]

Action

Out  \[ \boxed{\ldots} \quad ( \ldots ) \quad ( \ldots ) \]

SCION Router Verification Overview

Model

Environment Model
- attacker, network

Router Model

Reality

Real Environment

Router Code

Protocol
- Security Properties

Code
- Security Properties

satisfies
refined by
unproven
justified
proven

Verified SCION
Status

- Code verification tools built and prototyped
- First three levels of refinement completed
  - Improved understanding of protocols and properties
  - Uncovered numerous bugs and omissions
    - Revealed during modeling & formalization
    - Verified against implementation
- Next step: formally connect the two parts
Conclusions

- Internet, as designed, is insecure
- Scion architecture offers much stronger guarantees
- These can be put on a formal footing via refinement + code-level verification
- **Long term objective**: guaranteed back-door-free routers, made in Switzerland
We are hiring:  www.anapaya.net