Fault-Tolerant Forwarding in the Face of Malicious Routers

Alper T. Mızrak, Keith Marzullo, Stefan Savage
University of California, San Diego
Introduction

- Modern packet switched data networks.
- Data must be forwarded hop-by-hop from router to router towards destination.

- If a router is compromised, then
  - Control plane:
    - E.g. announce false route updates
  - Data plane:
    - E.g. alter, misroute, drop, reorder, delay or fabricate data packets.
- Threat model: Byzantine failures.
Goal

- Fault tolerant forwarding in the face of malicious routers.
- Detect the existence of compromised routers in a network.
- Remove them from the routing fabric.
Approach

- Given a routing protocol the decisions routers make are predictable.
  
  … so this problem is a candidate for *anomaly*-based intrusion detection

- A compromised router
  - can be identified by correct routers
  - when it deviates from exhibiting expected behavior.
Overview

- System Model
  - Network Model
  - Threat Model
- Traffic validation
  - Traffic summary functions
- Distributed detection
  - Specification
  - An example protocol
- Response: Changing the routing fabric
- Conclusion
Network Model

Assumptions

- The routing protocol provides each node with a global view of the topology:
  - Distributed link-state routing protocol: OSPF or IS-IS.
- Synchronous system:
  - Link-state protocols operate by periodically
    - measuring network connectivity
    - disseminating information
- Key distribution between pairs of nearby routers
- This overall model is consistent with the typical construction
  - Large enterprise IP networks
  - The internal structure of single ISP backbone networks
Definitions

- **Path**: a finite sequence of adjacent routers: \(<a, b, c, d>\)
- **X-path segment**: a sequence of x routers that is a subsequence of a path: \(<a, b, c>\), \(<b, c>\)

- A router is **faulty** if it introduces discrepancy into the traffic.
Threat Model

- **bad($k$):** Impose an upper bound on the number of adjacent faulty routers in any path.
  - **bad(2):** There can be no more than 2 adjacent faulty routers in any path.

- The routers at the source and sink of a flow are not faulty with respect to that flow's path.
Traffic Validation

- Way to tell that traffic isn’t disrupted en route
- We represent TV mechanisms as a predicate $\text{TV}(\pi, \text{info}_{r_i}^{\pi,\tau}, \text{info}_{r_j}^{\pi,\tau})$, where:
  - $\pi$ is a path segment $<r_1, r_2, \ldots, r_x>$
    - whose traffic is to be validated between routers $r_i$ and $r_j$
    - both $r_i$ and $r_j$ are in $\pi$;
  - $\text{info}_{r}^{\pi,\tau}$ is some abstract description of the packets
    - router $r$ forwarded
    - to be routed along $\pi$
    - over some time interval $\tau$.
- If routers $r_i$ and $r_j$ are not faulty, then
  - $\text{TV}(\pi, \text{info}_{r_i}^{\pi,\tau}, \text{info}_{r_j}^{\pi,\tau})$ evaluates to FALSE iff
  - $\pi$ contains a router that was faulty in $\pi$ during $\tau$. 
How to concisely represent \( \text{info}_r^{\pi,\tau} \)?

The most precise description of traffic
- an exact copy of that traffic.

Many characteristics of the traffic can be summarized far more concisely:
- Conservation of flow: a counter
- Conservation of content: a set of fingerprints
- Conservation of content order: a list of fingerprints
Traffic Validation Evaluation

- In an idealized network, TV might check $\text{info}_{r, \pi, \tau} = \text{info}_{r, \pi, \tau}$.
- However, real networks occasionally:
  - Lose packet due to congestion.
  - Reorder packets due to internal multiplexing.
  - Corrupt packets due to interface errors.
- TV must be sophisticated to accommodate this abnormal, but non-malicious behaviors.
- Implementing TV is a tricky engineering problem.
Overview

- **System Model**
  - Network Model.
  - Threat Model.

- **Traffic validation.**
  - Traffic summary functions.

- **Distributed detection.**
  - Specification.
  - An example protocol.

- **Response: Changing the routing fabric.**

- **Conclusion**
Specification

- A perfect failure detector (FD) would implement the following two properties:
  - Accuracy (tentative): An FD is Accurate if,
    - whenever a correct router suspects \((r, \tau)\),
    - then \(r\) was faulty during \(\tau\).
  - Completeness (tentative): An FD is Complete if,
    - whenever a router \(r\) is faulty at some time \(t\),
    - then all correct routers eventually suspect \((r, \tau)\) for some \(\tau\) containing \(t\).
Inadequate

- Implement the FD via Traffic Validation:
  - By collecting traffic information from different points in the network.

- Consider

- Any other router than $b$ and $c$
  - Can not distinguish between the case of $b$ being faulty and of $c$ being faulty.
  - Can only infer that at least one of $b$ and $c$ is faulty.
Weaken the Specification

- Detect suspicious *path segments*, not individual routers.
- An FD returns a pair \((\pi, \tau)\) where \(\pi\) is a path segment:
  - \(\alpha\)-Accuracy: An FD is \(\alpha\)-Accurate if,
    - whenever a correct router suspects \((\pi, \tau)\),
    - then \(|\pi| \leq \alpha\) and some router \(r \in \pi\) was faulty in \(\pi\) during \(\tau\).
  - \(\alpha\)-Completeness: An FD is \(\alpha\)-Complete if,
    - whenever a router \(r\) is faulty at some time \(t\),
    - then all correct routers eventually suspect \((\pi, \tau)\) for some path segment \(\pi\): \(|\pi| \leq \alpha\) such that
      - \(r\) was faulty in \(\pi\) at \(t\), and
      - for some interval \(\tau\) containing \(t\).
An Example Protocol: $\Pi_{k+2}$

A router $r$ has a set of path segments $P_r$ that it monitors.

- $P_r$ contains all the path segments that have $r$ at one end and whose length is at most $k+2$.
- $k$ is the maximum number of adjacent faulty routers along a path.

```plaintext
for each path segment $\pi$ in $P_r$:
   while (true) {
      synchronize with router $r'$ at other end of $\pi$;
      collect info$_{\pi,\tau}$ about $\pi$ for an agreed-upon interval $\tau$;
      exchange [info$_{\pi,\tau}$]$_r$ and [info$_{r',\pi,\tau}$]$_{r'}$ with $r'$ through $\pi$;
      if TV($\pi$, info$_{\pi,\tau}$, info$_{r',\pi,\tau}$) = FALSE then
         suspect $\pi$;
         reliable broadcast ($\pi,\tau$);
   }
```
Properties of Protocol $\Pi_{k+2}$

- $\Pi_{k+2}$ is $(k+2)$-Accurate:
- $\Pi_{k+2}$ is $(k+2)$-Complete.
  - If $r$ is faulty at some time $t$, then
    - $\exists$ a path segment $\pi$:
      - $r \in \pi$.
      - $r$ introduce discrepancy into the traffic through $\pi$ during $\tau$ containing $t$.
      - Only $f$ and $\bar{f}$-the first and last routers of $\pi$- are correct.
      - $3 \leq |\pi| \leq k+2$.
  - $f$ and $\bar{f}$ monitor $\pi$ and apply the $\Pi_{k+2}$ for $\pi$:
    - Compute TV ($\pi$, info$_{\pi,\tau}$, info$_{\bar{\pi},\tau}$) to be false
    - Suspect $\pi$, disseminate this information to the all other correct routers.
Overhead of Protocol $\Pi_{k+2}$

- This algorithm has reasonable overhead
  - For each forwarded packet compute a fingerprint.
  - Each router $r$ must synchronize and authenticate with the other end of each $\pi$ in $P_r$.
  - The size of $P_r$ dominates the overhead.
    - For Sprintlink network[Rocketfuel] of 315 routers and 972 links:
      - bad(1): a router monitors 35 path segments on average
      - bad(2): a router monitors 110 path segments on average
  - Dissemination of the suspected path segments can be integrated into the link state flooding mechanism.
Response

- What happens as a result of a detection?
- Need some *countermeasure* protocol.
  - Inform the administrator.
  - Immediate action:
    - Ideally would be part of the link state protocol.
    - We have a version of Dijkstra's SPF that can exclude suspected x-path segments.
Current Status

- We have implemented a prototype system, called *Fatih*.
- Runs in user-level on Linux cooperating with Zebra OSPF implementation.
- Gaining experience with traffic summary and validation mechanisms
- Still work-in-progress
Conclusion

- Specified the problem formally.
- Explore the implementation
  - Traffic validation
  - Distributed detection
  - Countermeasure
- It might be feasible to protect networks from insider attacks of routers.
- There's a lot of design and engineering to do first.
The end

- Thank you...
Challenges

- Efficient, compact and accurate traffic summaries
  - What disruptions can be detected?
  - Sensitivity vs overhead? Sampling?
- Traffic validation predicates
  - Noise: how to distinguish normal packet loss, reordering, corruption from malicious activities.
- Requires secure control plane.
WATCHERS

WATCHERS protocol developed (and criticized) at University of California, Davis through 2000.

Based on *conservation of flow*:

Input to a system must either be absorbed at that system or passed along to another system.
WATCHERS: Transit Packets

For each router pair \( x \) and \( y \), both routers maintain six counters for *transit packets*:

\[
D_{x,y} \quad T_{x,y} \quad S_{x,y} \\
S_{y,x} \quad T_{y,x} \quad D_{y,x}
\]
WATCHERS: Conservation of Flow

Conservation of flow can be expressed using these counters:

\[
\sum_{N:A \leftrightarrow N} \left( S_{N,A} + T_{N,A} \right) = \sum_{N:A \leftrightarrow N} \left( D_{A,N} + T_{A,N} \right)
\]
WATCHERS: Assumptions

System assumptions:

- Each router is a neighbor to at least one good router (*good neighbor* condition).
- Each pair of good routers has at least one path of only good routers connecting them (*good path* condition).
- A majority of the routers are good.
WATCHERS: Algorithm

Each router $A$ and each neighbor $N$:

   ... if they don’t agree, then $A$ diagnoses $N$ is bad.

2. Check counters of $N$ with those of each $N$’s neighbor $M$.
   ... if they don’t agree, then $N$ and $M$ will sort it out.
WATCHERS: Algorithm (cont'd)

3. Check conservation of flow with each neighbor $N$ using $N$’s counters.

\[ I = \sum_{\forall M: M \leftrightarrow N} (S_{M, N} + T_{M, N}) \]
\[ O = \sum_{\forall M: M \leftrightarrow N} (D_{M, N} + T_{M, N}) \]

If $|I - O| > T$ for some threshold $T$ then $A$ diagnoses $N$ as bad.
WATCHERS: Consorting Routers

By itself, this algorithm is not sufficient to detect consorting routers:

A sends to B

A → 1 → 2 → 3 → 4 → B

3 and 4 increment $D_{3,4}$ rather than $T_{3,4}$

4 discards

... so, changed algorithm so router maintains counters with each neighbor and destination (here, $B$).

Alper T. Mızrak; University of California, San Diego
WATCHERS: Discussion

Some observations:

- WATCHERS requires global synchronization for counter comparison.
- The *good neighbor* requirement is strong.
- Traffic validation is explicit.
- It took authors a few tries to get it right.
- There gave no real specification of the problem.