Isolating Compromised Routers

Alper Mizrak, Keith Marzullo and Stefan Savage
UC San Diego
Department of Computer Science and Engineering
Problem

- Routers are vulnerable points in the Internet, especially for insider attacks.
  - Attacks on the control plane versus the data plane.
- An attack could result in a router selectively altering, misrouting, dropping, reordering, delaying or fabricating data.

*Wish to isolate such routers from the network.*
Approach

- Routers make decisions whose effects can be observed.
- Given a routing protocol the decisions it makes are predictable.

... so this problem is a candidate for an intrusion detection-like approach.

- anomalous behavior.
Overview

This is work in progress (the Fatih project). Isolation for link state routing protocols.

- Present first solution to the problem (WATCHERS).
- High level summary of what we are doing.
- Thoughts about a methodology.

Fatih Sultan Mehmet II conquered Constantinople, the center of the Byzantine civilization.
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:

- $T_{x,y}$
- $D_{x,y}$
- $S_{x,y}$
- $T_{y,x}$
- $D_{y,x}$
- $S_{y,x}$
WATCHERS: Conservation of Flow

Conservation of flow can be expressed using these counters:

\[ \sum_{N: A \leftrightarrow N} (S_{N,A} + T_{N,A}) = \sum_{N: A \leftrightarrow N} (D_{A,N} + T_{A,N}) \]
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.

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

If $|I - O| > T$ for some threshold $T$ then $A$ diagnoses $N$ as bad.
By itself, this algorithm is not sufficient to detect *consorting routers*:

\[ A \to 1 \to 2 \to 3 \to 4 \to B \]

- 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, \( B \)).
WATCHERS: Discussion

- WATCHERS requires global synchronization for counter comparison; is this necessary?
- The *good neighbor* requirement is strong; is it necessary? If not, what is gained in its assumption?
- Traffic validation is embedded into the protocol.
- It took authors a few tries to get it right (and they still missed at least one case).
  - There gave no specification of the problem; only of their solution.
The problem of isolating routers has three components:

1. Collecting traffic information.
   - Monitoring is done on the basis of flows rather than links.
   - The type of data collected depends on the attacks that can be waged.
   - Sampling can be used to reduce the amount of data collected.
   - The amount of data collected also depends on what is done with it.
Fatih (II)

2. Disseminating data for detection to take place.
   - The system is synchronous (link state protocol) and, given a *good path* property, one can simply use consensus.
   - One can greatly reduce the amount of data collected and disseminated (but there’s a hitch).
3. Countermeasures when some router is found to be compromised.

- Fatih monitors flows, and so flows are removed rather than links being broken.
- Modified version of OSPF that excludes flows.
- Failure to effect a countermeasure can also be used as evidence of faulty router.
Nature of Specification

Cast as a *failure detector* problem that reports $x$-path segments as being faulty.

$x$-path segment: a path of length $x$.

Give an accuracy and completeness property for the failure detector of good routers.

- *Completeness*: as expected, but more complex.
- *$k$-accuracy*: $x$ is never larger than $k$.
  - High overhead: 2-accurate and complete.
  - Lower overhead: less accurate, essentially complete.
Restrictions

A router is faulty in $p$ if it can alter the data it forwards across $p$.

$\text{bad}(k)$: In any path $p$ no more than $k$ routers faulty in $p$ can be adjacent.

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

$\text{bad}(2)$, $s$ source, $t$ sink.
Current effort

Determine feasibility and overhead (space and time).

- Implementation into access router.
- Simulation using Rocketfuel-determined ISP topologies.
Methodology Questions (I)

This is (by far) not the first attempt to apply fault-tolerance techniques to make systems more secure.

- Treating an intruded system as one that can exhibit arbitrary failures seems well-founded.
- Other methods from fault tolerance aren’t so well founded (e.g. threshold assumptions).
Methodology Questions (II)

- When designing a fault-tolerant protocol, one uses the *system model*: a set of assumptions about the environment (eg, bad($k$), synchronous communications).

- A correct protocol often relies on specific properties of the system model.
  - Eg, a $t+1$ round protocol for Consensus in an asynchronous environment.
Methodology Questions (III)

- When deploying a protocol, one instantiates the system model to attain a target level of availability.
  - Eg, use MTTF/MTTR and target availability to compute a desired value of $t$.

- In practice, a well-engineered protocol checks to see if it is in a state that implies the system model doesn’t hold.
  - Eg, if in round $t$ of Consensus, stop rather than deciding.
  - The beauty of protocols like Paxos and $\langle \rangle$S consensus is that they automatically do this.
Methodology Questions (IV)

- When one is concerned with security (as compared with fault tolerance), system model assumptions are called *threats* and are subject to attack.

- An essential design aspect of secure systems is ensuring that the threats can be “monitored” (eg, audit logs, IDS).

- This is the essence of the concern about using threshold-based protocols for security purposes.
Methodology Questions (V)

- SIFT adopted the threshold model as an abstraction the engineers could use.
- Need a similar exercise to develop a methodology of applying fault-tolerance techniques to make systems secure.
  - Eg, is bad($k$) reasonable?
  - Studying BFT.