Detecting Compromised Routers via Packet Forwarding Behavior

Alper T. Mizrak, VMware
Stefan Savage and Keith Marzullo, UCSD

Abstract

While it is widely understood that criminal miscreants are subverting large numbers of Internet-connected computers (e.g., for bots, spyware, SPAM forwarding), it is less well appreciated that Internet routers are also being actively targeted and compromised. Indeed, due to its central role in end-to-end communication, a compromised router can be leveraged to empower a wide range of direct attacks including eavesdropping, man-in-the-middle subterfuge, and denial of service. In response, a range of specialized anomaly detection protocols has been proposed to detect misbehaving packet forwarding between routers. This article provides a general framework for understanding the design space of this work and reviews the capabilities of various detection protocols.

It is widely understood that the Internet is awash in threats. The mean time for a vulnerable system to be infiltrated once connected to the Internet is typically measured in minutes. Consequently, most research effort in network security has focused on protecting these end systems or mitigating the impact of their compromise. However, the Internet architecture relies equally strongly on the correct behavior of the intermediate routers that forward packets, hop by hop, to their eventual destination. While it is not as widely appreciated, these network routers are also under attack.

For example, at the 2005 Black Hat Briefings, Mike Lynn demonstrated how Cisco routers can be compromised via simple software vulnerabilities. Even simpler, others have documented how combinations of social engineering and weak passwords can and have been used to compromise thousands of Internet routers, and there is an underground market for trading access to them [1–4]. Finally, in their annual surveys of the Internet network security operations community, Arbor Networks Inc. have repeatedly documented such attacks across a range of Internet service providers (ISPs) [5].

Once a router has been compromised, the standard command line interface from vendors such as Cisco and Juniper can be used to selectively drop packets, delay them or transparently "tunnel" through arbitrary-third party hosts and back again [6, 7]. Together these capabilities are sufficient to selectively eavesdrop, deny or degrade service, or construct a man-in-the-middle attack against any host that receives service through the router.

Thus, a number of researchers have recently explored the problem of detecting such router compromises as revealed from their inconsistent packet forwarding behavior. In this article we provide a general framework for understanding this work and then study some of the detection protocols [8–11].

This material is based upon his dissertation research at UCSD.

Centralized Failure Detector via Active Replication

The expected behavior of a router is predictable: traffic enters a router and is forwarded on to the next hop toward its destination. Thus, the behavior of a router can be verified by a failure detector via an identical replica of that router.[1] For example, in Fig. 1, a failure detector is implemented with an identical replica r' of the router r. In this scheme the failure detector observes router r's input traffic and delivers an identical stream of packets to router r'. In turn, the failure detector verifies that the output traffic exiting router r' is identical to that leaving router r. If there is a discrepancy, an alarm is raised indicating that either the monitored router is faulty or the failure detector is faulty. This is an ideal failure detector for revealing any anomalous forwarding behaviors. However, this approach has a number of practical limitations that make it infeasible to deploy at scale.

Complexity of Implementation

First of all, maintaining state consistency between a router and its replica can be highly demanding to engineer. For example, a router will receive routing updates that cause its forwarding tables to be recomputed. If these forwarding tables are not updated at the same moment on both the router and its replica, a temporary traffic forwarding inconsistency may occur. Similarly, a number of active queue management schemes make use of randomization to drive their operation. Thus, to faithfully replicate a router's behavior also requires replicating its random number stream.

[1] *This scheme is also called master-checker, active replication, or state machine approach in the literature.*

0890-8044/08/$25.00 © 2008 IEEE

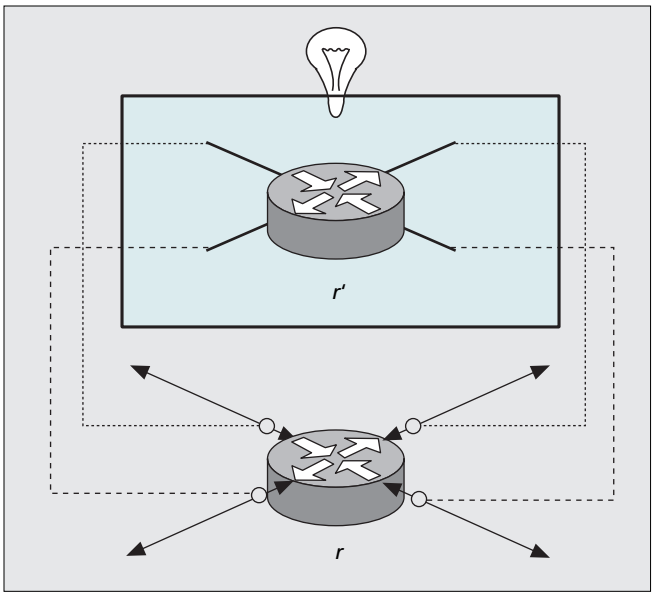

Figure 1. *Failure detector via the active replica/state machine approach.*

Resource Requirement

Furthermore, to implement a replica-based failure detector, as in Fig. 1, requires additional hardware resources — an identical replica of each router — and additional communication channels — the passive monitors on each router's input and output links. These can be prohibitively expensive.

Researchers have tried to address these issues in two ways. First, they have developed weaker but lighterweight failure detectors via *traffic validation*: instead of validating the exact traffic that transits a router, various characteristics of the traffic entering and leaving various parts of the network can be used for validation. Second, they have developed distributed failure detectors that exploit the participation of correct routers rather than requiring a separate monitoring infrastructure. We frame each of these approaches in turn below.

Traffic Validation

Traffic validation is the basis for detecting anomalous behavior: given traffic entering a region of the network, and knowing the expected behavior of the routers in the network, anomalous behavior is detected when the monitored traffic leaving that part of the network differs significantly from what is expected. Traffic validation can be defined in terms of *conservation of traffic*.

Conservation of Traffic

Some *property* of the *traffic* entering a *region of a network* must be consistent with the same property of the traffic leaving that part of a network.

There are three design decisions that must be addressed to implement such a mechanism.

Which Property of the Traffic Is to Be Validated? — Upon receiving a packet, a router references its routing table to determine the next hop toward the destination, and then forwards the packet. Thus, ideally, the traffic entering a router is equal to the traffic leaving that router. Of course, packets experience queuing and processing delay, and packets can be lost due to congestion. The most precise description of traffic is itself — the exact content of the packets. However, the storage requirements to buffer all packets (as well as the bandwidth consumed by resending them to implement distributed detection, discussed later) make this approach impractical. Instead, one can validate more concise properties that approximate this ideal. Some such properties are:

- Conservation of *flow* validates the volume of the traffic, thereby detecting maliciously dropped packets.
- Conservation of *content* validates the content of the traffic, thereby detecting maliciously modified packets.
- Conservation of *order* validates the order among the packets that constitute the traffic, thereby detecting maliciously reordered packets.
- Conservation of *timeliness* validates the time behavior of the forwarding process, thereby detecting maliciously delayed packets.

What Traffic Is to Be Monitored? — The protocols can be categorized in various ways. Some protocols monitor a single packet, while others monitor aggregate traffic. Some protocols are based on active probing: they send probe packets periodically, while others deploy a passive approach that simply monitors existing traffic.

What Region of a Network Is to Be Monitored? — Various existing protocols apply conservation of traffic at different granularities in a network, including per interface, per router and per path segment.

A failure detector based on traffic validation can be almost as effective as one based on active replication, but their reduced overhead makes them far more practical. Thus, all the detection protocols examined in this article are based on traffic validation, each representing a slightly different point in the design space. In practice, designing a traffic validation mechanism includes engineering trade-offs for each design decision above. For example, real networks occasionally lose packets due to congestion; thus, a traffic validation mechanism must distinguish between benign congestive loss and packet losses caused by malicious actions.

Distributed Detection

Instead of a centralized failure detector, as in Fig. 1, most systems approximate this service via a distributed protocol that relies on the participation of uncompromised routers. Underlying this approach is the observation that while a compromised router can forward traffic any way it wishes, other routers have an expectation of how traffic should be forwarded. Thus, as long as traffic traverses neighboring uncompromised routers as well, there will be sufficient observations to detect anomalous behavior. Traffic validation systems implement such a distributed failure detector by synchronizing traffic information between two or more routers and then redistributing this information to allow for detection.

Refining this model, a compromised router can be *traffic faulty* by forwarding traffic in a faulty manner, as well as *detection protocol faulty* by behaving arbitrarily with respect to the detection protocol. Both cases should be considered. Also, more than one router may be compromised at a time. We assume that there is sufficient path diversity between any two uncompromised routers such that they can communicate without traversing a compromised router. In a sense this assumption is pedantic, since this assumption is needed to ensure that the communication network is not partitioned, and it is impossible to guarantee any network communication across a partition.

We also assume that a host's access router is not compromised. This is a strong assumption, but necessary: if a host's access router is compromised, the host is partitioned from the rest of the network, and there is no routing remedy even if an

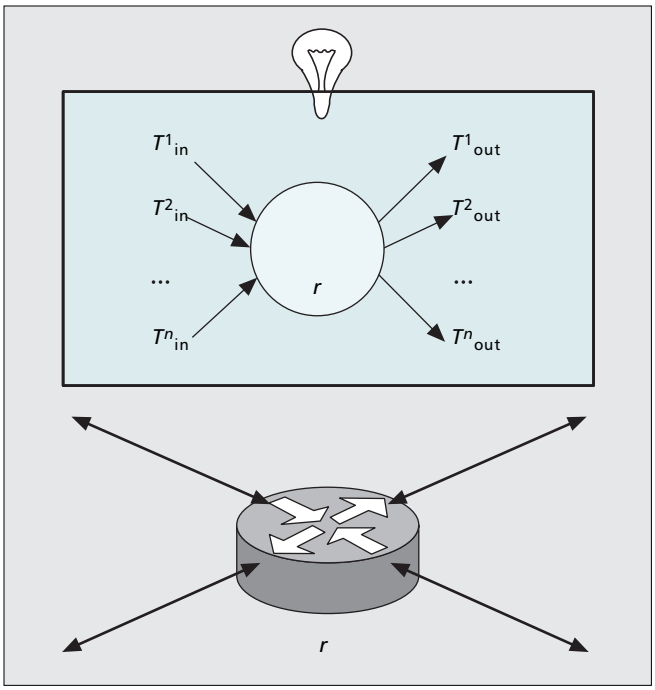

Figure 2. *Failure detector via a traffic validator per router.*

anomaly is detected. Moreover, from the standpoint of the network such traffic originates from a compromised router, and therefore cannot demonstrate anomalous forwarding behavior. In effect, distributed detection protocols can only hope to detect anomalous routing between any two pairs of uncompromised nodes.

There is inherently some uncertainty in failure detection since one cannot generally distinguish between a true failure and a failure in the failure detector itself (this is true even with the idealized active replication scheme). This abstract limitation manifests in distributed traffic validation protocols as well. For example, suppose router r_1 collects traffic information about packets that traverse routers r_1 to r_2 to r_3. Based on the information r_3 collects, suppose r_3 determines that packets have been dropped between r_1 and r_2. Unfortunately, r_3 cannot distinguish between the case of r_1 lying about what packets it sent to r_2 and r_2 lying about what packets it received from r_1. Hence, there is an inherent lack of precision in determining which routers are compromised.

We have failure detectors report suspicions as *path segments*, where a path segment is a sequence of consecutive routers that is a subsequence of a path.[2] More specifically, a failure detector reports a path segment if it suspects a router in that path segment isbehaving in a faulty manner.

This section is a brief summary of the formal specification presented in [12]. We refer readers to [12] for the derivation of the formal specification. In short, we cast the problem as a failure detector with *completeness*, *accuracy,* and *precision* properties.

Completeness

Whenever a router forwards traffic in a faulty manner:

- If *all correct routers* eventually suspect that a path segment contains a faulty router, a failure detector is *strong-complete*.
- If *at least one correct router* eventually suspects that a path segment contains a faulty router, a failure detector is *weak-complete*.

Accuracy

A failure detector is *accurate* if, whenever a correct router suspects a path segment, there is at least one faulty router in that path segment.

Precision

A failure detector also has a *precision*, which is the maximum length of a path segment it suspects.

To be useful, a failure detector must be complete and accurate, and one would prefer a smaller precision. Implementing such distributed detection involves trade-offs among precision, completeness, and the overhead of monitoring and communication. Various detection protocols address these design decisions in different ways, as we discuss in the next section.

Compared to weak completeness, strong completeness is more desirable since every correct router detects the fault. Given a weak-complete detector, a strong-complete one can be implemented, but it may not be simple and some precision would be lost. For example, consider that a source router r_s detects a link $\langle r_1, r_2\rangle$ as faulty. Announcing this detection, the other correct routers in the network have to consider the case that r_s is faulty as well. On the other hand, in some cases having a weak-complete detector is enough for making a proper response. For example, relying on source routing, a router detecting a failure just computes a new route excluding the suspected path segment. In the above example, r_s may only update its own routing table excluding the suspected $\langle r_1, r_2\rangle$.

Case Studies

In this section we describe existing protocols proposed to detect and mitigate attacks on the network data plane. We examine both the trade-offs in their choice of traffic validation mechanisms and the particular distributed protocols used to disseminate this information.

Traffic Validation Per Router: WATCHERS

The most similar approximation to the failure detector in Fig. 1 is WATCHERS [8], which detects and isolates faulty routers based on a distributed network monitoring approach. A faulty router is defined as one that drops or misroutes packets, or behaves in an arbitrary manner with respect to the proposed protocol. A conservation of flow principle (CoFP) was proposed to detect faulty routers. Basically, CoFP states that the amount of traffic entering a router should be equal to the amount of traffic leaving that router.

The traffic validator WATCHERS implements is given in Fig. 2 as a centralized service. WATCHERS validates the CoFP of the aggregate traffic entering into each router in the network. If the difference between the volume of the traffic entering and leaving the router exceeds a user-defined threshold, a failure is detected and an alarm is raised. This threshold is needed to avoid false positives as a result of congestive packet losses.

WATCHERS implements this failure detector by requiring all the neighboring routers of a router r to synchronize with each other, count how many bytes they have received from and forwarded to r during an agreed-upon time interval, distribute snapshots of their counters to the others by flooding, and finally, validate the CoFP.

If a neighbor router r_n can not validate the router r, r_n announces that the link $\langle r_n, r\rangle$ is suspicious, and $\langle r_n, r\rangle$ is removed from the routing fabric.

[2] *For example, if a network consists of the single path* $\langle r_1, r_2, r_3, r_4\rangle$, $\langle r_2, r_3\rangle$ *is a path segment, but* $\langle r_1, r_3\rangle$ *is not because* r_1 *and* r_3 *are not adjacent.*

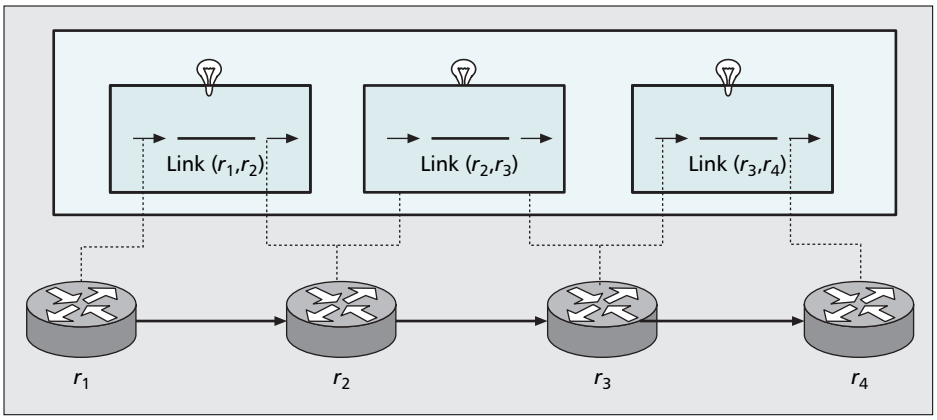

Figure 3. *Failure detector via a traffic validator per path-segment nodes.*

In terms of our specification, WATCHERS is *accurate* with a *precision* of 2. It is not *complete* as described, but can be modified to be *strong-complete* as explained in [13].

Generally speaking, there are two limitations of WATCHERS:

- The main drawback of WATCHERS is its restrictive threat model: it can only detect malicious packet drops and misroutes. Several researchers [9–11] have subsequently developed protocols with more general traffic validation mechanisms addressing a more comprehensive set of attacks.
- The architects of WATCHERS noticed a completeness problem caused by "consorting faulty routers," a problem first described by Perlman [14]. Faulty routers are said to be consorting if they launch a coordinated attack and cooperate to *hide each other's malicious behavior*. The WATCHERS protocol addressed this issue by requiring each router to maintain state for every neighbor and destination pair in the network. Other protocols have addressed this problem with a different approach: validating traffic over path segments. We discuss this next.

Traffic Validation per Path Segment Nodes: HSER

Avramopoulos *et al.* [9] present a protocol called Highly Secure and Efficient Routing (HSER), a combination of source routing, hop-by-hop authentication, a priori reserved buffers, sequence numbers, timeouts, end-to-end reliability mechanisms, and fault announcements. While none of these individual mechanisms is novel by itself, it is the combination of them that delivers Byzantine robustness and detection.

The traffic validator HSER implements is given in Fig. 3 as a centralized service. HSER validates the conservation of content property of a single packet that is monitored along the path from the source to the destination. If any router along the path discovers that its neighbor has lost or altered the packet, a failure is detected and an alarm is raised.

HSER implements its failure detector by requiring each router along the path to compute a fingerprint for the monitored packet, keep a timeout, and finally, validate the conservation of content property with its neighbors. Upon receiving a packet, the router first validates the authenticity and forwards the packet to the next hop toward the destination. After forwarding the packet, the router sets a timeout to the worst case round-trip time to the destination from itself. If the authenticity of the packet is not verified or the timeout expires, the router generates a *fault announcement*, including its neighbor and itself, to send back to the source.

HSER relies on source routing to act on its findings. Thus, upon receiving a fault announcement, the source router computes a new route to the destination excluding the detected link from its routing fabric.

In terms of our specification, HSER is *weak-complete* —since only the source detects a failure — and *accurate* with a *precision* of 2. Other protocols based on this approach are presented in [12, 15, 16].

However, the overhead of this approach is quite high, since for every source and destination pair, all of the routers along the path must participate in the detection protocol. To mitigate this overhead, other researchers proposed two complementary optimizations:

- Give up precision by only validating at the end routers of a *path segment* (i.e., *a contiguous sequence of routers on the path*), in which case none of the intermediate routers along the path participates in detection. However, it also becomes impossible to isolate faults to a finer granularity than the length of the path segment being monitored.
- Give up accuracy by generating a sampling schedule and only sampling the chosen packets at end routers of the path segment. While this method significantly decreases overhead, attacks on unsampled packets cannot be detected.

Traffic Validation Per Path Segment Ends: SecTrace

SecTrace [10] was developed to be a practical tool for securely tracing the path of traffic from a source toward a particular destination. It proceeds hop by hop similar to traceroute: at each round, the source validates the traffic between itself and an intermediate router toward the destination.

The traffic validator SecTrace implements is given in Fig. 4 as a centralized service. SecTrace validates the conservation of content property of the aggregate or sampled traffic[3] between the source router and an intermediate router. If the source detects a discrepancy in the traffic, a failure is detected and an alarm is raised.

SecTrace implements such a failure detector distributed in the network by requiring only the end routers of the monitored path segment to synchronize with each other and to compute fingerprints for the traffic between themselves for an agreed-upon time interval. At the end of a round, the corresponding intermediate router sends back the information it has collected and the identity of the next expected router

[3] *It can adopt both active probing and passive monitoring approaches.*

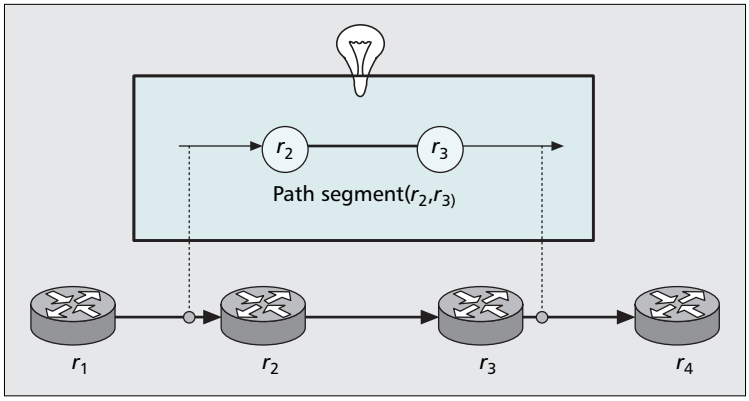

Figure 4. *Failure detector via a traffic validator per path-segment ends.*

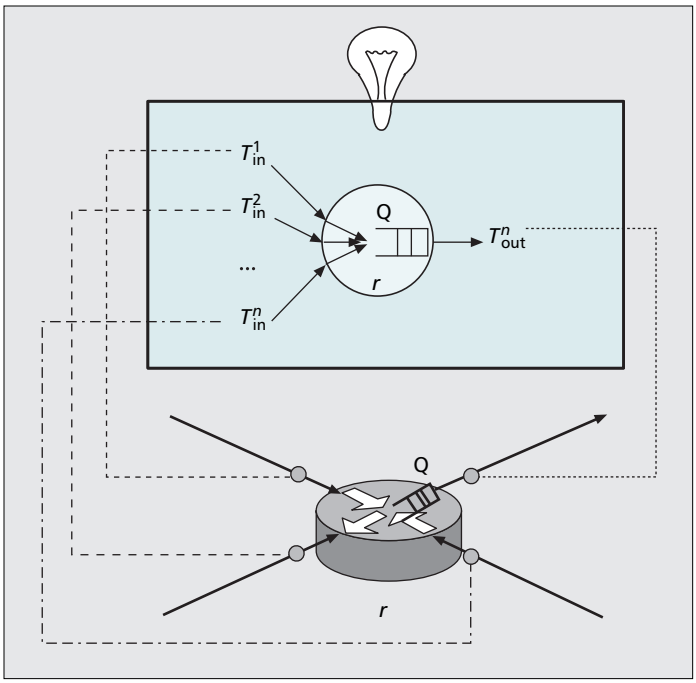

Figure 5. *Failure detector via a traffic validator per interface.*

toward the destination. Upon receiving this information, the source router validates the conservation of content property: if the source validates the traffic, it initiates another `SecTrace` round with the next intermediate router toward the destination; otherwise, the source detects a failure.

For example, in Fig. 4 a path segment of $\langle r_1, r_2, r_3, r_4\rangle$ is monitored during the given traffic validation round, and only the source r_1 and the corresponding intermediate router r_4 implement the distributed failure detector. If r_1 detects a discrepancy in the traffic, a failure is detected and an alarm is raised. Either one of the intermediate routers $\{r_2, r_3\}$ is traffic faulty, introducing discrepancy into the monitored traffic, or the failure detector, which is implemented by r_1 and r_4, is detection protocol faulty: at least one of $\{r_1, r_4\}$ is faulty.

In terms of our specification, `SecTrace` is *weak-complete* — since only the source detects a failure — and *accurate* with a *precision* of k, where k is the length of the monitored path segment.

However, this accuracy is predicated on the assumption that malicious actions are consistent over time and not Byzantine. For example, in Fig. 4, if r_1 could not validate the traffic with r_4, it would detect $\langle r_3, r_4\rangle$ as faulty. The reasoning behind this approach is that r_1 was able to validate the same traffic up to the upstream neighbor r_3 in the previous validation round, so either r_3 or r_4 must be faulty, introducing traffic discrepancy. However, if faulty routers can modify their behavior, this assumption violates the accuracy property. For example, assume that router r_2 is faulty, but only alters traffic *after* r_1 completes a validation round with r_3. Consequently, $\langle r_3, r_4\rangle$ would be identified as faulty, while both r_3 and r_4 are correct. To address this problem, the authors propose to give occasional indications of `SecTrace` activity, such as by continuously sending round initialization packets pretending to monitor the traffic, while in reality doing nothing.

Finally, they propose three different countermeasures in response to a detected failure:

- The source tries to route the traffic around the detected link using source routing.
- The source notifies the downstream routers, expecting them to make the appropriate routing adjustments avoiding the suspected routers.
- The source alerts the administrator of the suspected routers.

Other protocols based on this approach are presented in [12, 17–21].

Traffic Validation per Interface: `Protocol` χ

Unfortunately, it is quite challenging to attribute a missing packet to a malicious action because modern IP networks routinely drop packets when the load temporarily exceeds a router's buffering capacity. Almost all detection protocols have tried to address this problem using a user-defined threshold. Unfortunately, using such a threshold will necessarily create unnecessary false positives or mask highly focused attacks. One exception is `Protocol` χ [11], which dynamically infers the number of congestive packet losses that will occur based on measured traffic rates and buffer sizes. Once the ambiguity from congestion is removed, subsequent packet losses can be attributed to malicious actions based on one of the previous traffic validation mechanisms.

The traffic validator `Protocol` χ implements is given in Fig. 5 as a centralized service. `Protocol` χ validates the conservation of timeliness property for aggregate traffic entering each interface of a router in the network. If a packet loss occurs when the monitored interface's queue is not predicted to be full, a failure is detected and an alarm is raised.

`Protocol` χ implements such a failure detector distributed in the network by requiring each neighbor of a router to synchronize with each other, compute a fingerprint with a timestamp for each packet that passes through the interface Q during an agreed-upon time interval, distribute these information among the other neighboring routers, and finally, validate the conservation of timeliness property of the traffic by simulating the behavior of the monitored interface queue.

If a neighbor router r_n detects that router r drops a packet when the corresponding queue is not full, r_n announces that link $\langle r_n, r\rangle$ is suspicious, and $\langle r_n, r\rangle$ is removed from the routing fabric.

In terms of our specification, `Protocol` χ is *strong-complete* and *accurate* with a *precision* of 2.

`Protocol` χ can be extended to address the problem of adjacent consorting faulty routers by monitoring every output interfaces of the neighbors k hops away and disseminating the traffic information to all neighbors within diameter hops. However, this approach can also add significant additional communications overhead when networks are densely connected (as described in [12]). The detection protocol in [22] is also based on this approach.

Conclusion

In this article we have described a general framework for understanding the literature on detecting malicious routers via packet forwarding behavior. We have described how traffic validation is the basis for all such schemes, and distributed per-router validation is a simple approximation to an idealized detector. However, due to the threat of consorting faulty routers (i.e., multiple routers on a path have been compromised), practical systems are limited to validating path segments rather than individual routers. Within this approach there is a trade-off between precision and overhead — depending on the span of a segment — and most protocols have chosen to explore the lower overhead part of this design space. Finally, all protocols are subject to noise due to congestive packet loss, which is difficult to distinguish from malicious dropping. While per-interface

techniques can differentiate between these conditions, it comes at the expense of high overhead. For in-depth analysis of these protocols and further issues, the readers are referred to [23].

In a short time there have been significant advances in this domain, and while none of these protocols has yet been deployed in a production network, they are quickly becoming cheap enough and precise enough to be a viable option against router-oriented attacks.

References

[1] R. Thomas, "ISP Security BOF, NANOG28," June 2003, http://www.nanog.org/mtg-0306/pdf/thomas.pdf
[2] X. Ao, "DIMACS Report: Workshop on Large Scale Internet Attacks," Nov. 2003.
[3] M. Wolfgang, "Exploiting Cisco Routers," Sept. 2003, http://www.securityfocus.com/infocus/1734
[4] M. Aharoni and W. M. Hidalgo, "Cisco SNMP Configuration Attack with a GRE Tunnel," Sept. 2005, http://www.securityfocus.com/infocus/1847
[5] D. McPherson and C. Labovitz, "Worldwide Infrastructure Security Report," Sept. 2006, http://www.arbornetworks.com/sp\security\report.php
[6] Gauis, "Things to Do in Ciscoland When You're Dead," Jan. 2000, http://www.phrack.org
[7] D. Taylor, "Using a Compromised Router to Capture Network Traffic," unpublished tech. rep., July 2002.
[8] K. A. Bradley *et al.*, "Detecting Disruptive Routers: A Distributed Network Monitoring Approach," *Proc. IEEE Symp. Sec.and Privacy*, May 1998, pp. 115–24.
[9] I. Avramopoulos *et al.*, "Highly Secure and Efficient Routing," *INFOCOM '04*, Amendment, Feb. 2004
[10] V. N. Padmanabhan and D. R. Simon, "Secure Traceroute to Detect Faulty or Malicious Routing," *SIGCOMM Comp. Commun. Rev.*, vol. 33, no. 1, 2003, pp. 77–82.
[11] A. T. Mizrak, K. Marzullo, and S. Savage, "Detecting Malicious Packet Losses, ' UCSD, tech. rep. CS2007-0889, Apr. 2007.
[12] A. T. Mizrak *et al.*, "Detecting and Isolating Malicious Routers," *IEEE Trans. Dependable and Secure Computing*, vol. 3, no. 3, July.–Sept. 2006, pp. 230–44.
[13] A. T. Mizrak, K. Marzullo, and S. Savage, "Detecting Malicious Routers," UCSD, tech. rep. CS2004-0789, May 2004.
[14] R. Perlman, "Network Layer Protocols with Byzantine Robustness," Ph.D. dissertation, MIT LCS TR-429, Oct. 1988.
[15] K. Argyraki *et al.*, "Providing Packet Obituaries," *Proc. ACM SIGCOMM HotNets-III*, 2004.
[16] A. Herzberg and S. Kutten, "Early Detection of Message Forwarding Faults," *SIAM J. Comp.*, vol. 30, no. 4, 2000, pp. 1169–96.
[17] B. Awerbuch*et al.*, "An On-Demand Secure Routing Protocol Resilient to Byzantine Failures," *ACM Wksp. Wireless Sec.*, Sept. 2002.
[18] I. Avramopoulos and J. Rexford, "Stealth Probing: Efficient Data-Plane Security for IP Routing," *Proc. USENIX Annual Tech. Conf.*, May–June 2006.
[19] S. Lee, T. Wong, and H. S. Kim, "Secure Split Assignment Trajectory Sampling: A Malicious Router Detection System," *Proc. IEEE DSN '06*, June 2006, pp. 333–42.
[20] D. Wendlandt *et al.*, "Don't Secure Routing Protocols, Secure Data Delivery," *Proc. 5th ACM Wksp. Hot Topics in Networks* Nov. 2006.
[21] S. Goldberg *et al.*, "Measuring Path Quality in the Presence of Adversaries: The Role of Cryptography in Network Accountability," Princeton Univ., tech. rep., 2007.
[22] W. Zhang *et al.*, "Secure Routing in Ad Hoc Networks and a Related Intrusion Detection Problem," *IEEE MILCOM*, 2003.
[23] A. T. Mizrak, "Detecting Malicious Routers," Ph.D. dissertation, UCSD, Sept. 2007.

Biographies

ALPER T. MIZRAK [SM] (amizrak@cs.ucsd.edu) received a B.S. degree in computer engineering from Bilkent University, Ankara, Turkey, in 2000 and a Ph.D. degree in computer science from the University of California, San Diego, in 2007. He is a senior member of technical staff at VMware. His research interests are computer networks and fault-tolerant distributed systems.

STEFAN SAVAGE [M] received a B.S. degree in applied history from Carnegie-Mellon University and a PhD degree in computer science and engineering from the University of Washington. He is an associate professor of computer science and engineering at the University of California, San Diego, and currently serves as director of the Cooperative Center for Internet Epidemiology and Defenses (CCIED), a joint project between UCSD and the International Computer Science Institute. His research interests lie at the intersection of operating systems, networking, and computer security.

KEITH MARZULLO [M] received a Ph.D. degree from Stanford University in 1984. He is a professor in the Computer Science and Engineering Department at the University of California, San Diego where he has been since 1993. He works on both theoretical and practical issues of fault-tolerant distributed computing in various domains, including grid computing, mobile computing, and clusters.