Towards Self-Explaining Networks

Wenchao Zhou[⋆] Qiong Fei[⋆] Andreas Haeberlen[⋆] Boon Thau Loo[⋆] Micah Sherr[◇]
[⋆]*University of Pennsylvania* [◇]*Georgetown University*

Abstract

In this paper, we argue that networks should be able to explain to their operators *why* they are in a certain state, even if – and particularly if – they have been compromised by an attacker. Such a capability would be useful in forensic investigations, where an operator observes an unexpected state and must decide whether it is benign or an indication that the system has been compromised. Using a very pessimistic threat model in which a malicious adversary can completely compromise an arbitrary subset of the nodes in the network, we argue that we cannot expect to get a complete and correct explanation in all possible cases. However, we also show that, based on recent advances in the systems and the database communities, it seems possible to get a slightly weaker guarantee: for any state change that directly or indirectly affects a correct node, we can *either* obtain a correct explanation *or* eventually identify at least one compromised node. We discuss the challenges involved in building systems that provide this property, and we report initial results from an early prototype.

1 Introduction

Operators of networks often find themselves needing to answer a diagnostic or forensic question. As an illustrative example, we consider the scenario where a deployed network is found to be in an unexpected state: a suspicious routing table entry is discovered or a proxy cache is found to contain an unusually large number of entries. The operator must determine the *causes* of this state before he can decide on an appropriate response. On the one hand, there may be an innocent explanation: the routing table entry may be caused by a misconfiguration, or the cache entries may simply be the result of a workload change. On the other hand, the unexpected state may be the symptom of an ongoing attack: the routing table entry may be the result of route hijacking, and the cache entries may be a side-effect of a malware infection. If the network is indeed under attack, the operators must act quickly to prevent further damage.

Once an attack or intrusion is discovered, a different challenge arises. Suppose the operators have discovered that a certain set of nodes has been compromised. To repair the damage, it may be insufficient to disinfect these specific nodes, since the damage may have already spread to the rest of the network. For example, the adversary may have already polluted routing tables on other nodes, which may allow the adversary to continue her control on the network traffics even after the compromised node is taken offline. Restoring a backup of the entire network would solve this problem, but such a solution is disruptive and might cause a considerable amount of work to be lost. It would be more preferable if the operators could determine the precise *effects* of the intrusion.

In this paper, we ask a very simple question: *Is there a way to build networks such that they can explain their own state, even if (and especially if) they are under attack?* Networks of this type, which we refer to as *self-explaining networks*, could help with both of the tasks we have motivated above. To solve the first task, the operator could simply ask: 'Why does the routing table on node X contain an entry Y?'. The network might then respond, for example, by explaining that the entry was created as a result of a configuration change on another node Z, which caused a previously blocked route to be propagated to X. From this, the operator might learn that an unauthorized person had accessed the console on Z at time T. He could then solve the second task by asking what other changes had been made on Z around time T, and how they had (transitively) affected the rest of the network. He could then identify the affected nodes and undo any damage that had been done to them.

Designing a self-explaining network involves at least two major challenges. First, it is not at all obvious how such 'why'-questions can be formulated, or what the answer should look like. Second, it is not immediately clear that the system can even generate a useful answer. This is because the most important questions are inherently asked at the most inconvenient time, namely when the system is already under attack. The more severe the attack, the more important a correct answer becomes! A simple portscan or a failed `sudo` is easy to detect but also not particularly dangerous. The moment of truth

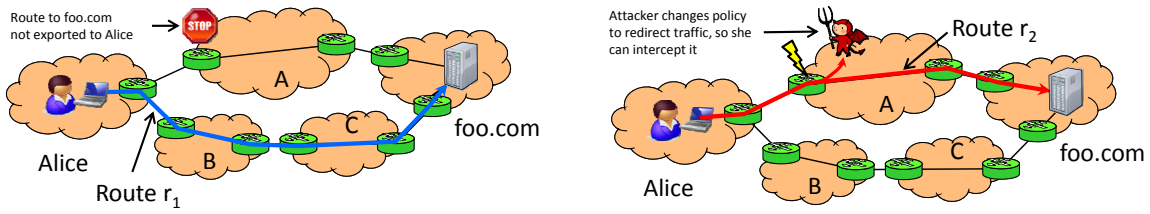

Figure 1: Example scenario. In the unmodified system (left), network A's policy does not allow cross traffic, so Alice can reach `foo.com` only through networks B and C. If evil Eve can compromise A's router (right), she can change this policy and thus gain the ability to listen to Alice's traffic.

comes when an attacker is exploiting a bug in the hypervisor or is circumventing the trusted hardware module, or when a malicious insider has installed malware on the majority of the nodes, which are now providing incorrect data in order to confuse the operator. Attacks of this magnitude are rare, but they do occur [10, 16, 23], and even for less spectacular attacks it is difficult to anticipate which of the (typically many) assumptions behind a network design the attacker may be able to violate.

In this work, our goal is to build a self-explaining network that can function even during very severe attacks. Hence, we assume that the adversary can gain complete control over the nodes he has compromised. This is a major difference to prior work, which typically assumes some kind of trusted component, such as a trusted kernel [11, 17], a trusted VMM [1], a trusted infrastructure [22, 24], or trusted hardware [4]. A network that does not rely on such trusted components should be harder to attack successfully; however, it will also be more difficult to build.

Nevertheless, there is evidence that it may be possible to build practical self-explaining networks. On the one hand, the systems community has recently developed techniques for fault detection that require only minimal assumptions; techniques like PeerReview [7] work in the presence of arbitrary Byzantine faults and do not require a bound on the number of compromised nodes or trust in any hardware or software on such nodes. On the other hand, the database community has been working on data provenance [3, 25], which provides a way to formulate 'why'-questions in such a way that they can be processed and answered automatically.

By itself, neither technology is sufficient to solve the problem. PeerReview, for example, can detect certain types of faults automatically – specifically, incorrect state transitions – but it is not effective against other fault classes, such as instabilities arising from interactions between multiple nodes, and it cannot explain the problem or determine its effects on other nodes. Existing data provenance techniques, on the other hand, can only answer 'why'-questions *in the absence of faults or attacks*; they cannot detect when faulty nodes forge incorrect responses. Although secure provenance techniques do exist [8], they are designed for application data and cannot handle common forms of misbehavior in networks, such as equivocation. Finally, as we will explain in Section 3, it is also not sufficient to simply layer a provenance system on top of a fault detection system, since the adversary can exploit interactions between the two layers to conceal an attack. This problem is fundamental and not merely a consequence of a few bugs; for example, the existing provenance models need to be extended and refined before they can be safely used in an adversarial setting.

Under the very pessimistic threat model we consider here, it seems infeasible to guarantee that the system will completely and correctly answer queries in *all* possible circumstances. However, it is possible to guarantee something slightly weaker: for any state change that directly or indirectly affects at least one correct node, we can *either* obtain a correct explanation *or* eventually identify where the compromised nodes have lied. Since the operator learns something useful in both cases, this seems like an attractive property for a practical system.

In the rest of this paper, we describe our vision of self-explaining networks that is based on a combination of these two technologies. We outline the challenges beyond these two technologies that will have to be addressed to make this vision a reality – such as adapting the provenance models from databases for use in a potentially maclicious network, developing suitable mechanisms for extracting and tracking provenance, formalizing and proving the guarantees to be offered by self-explaining networks, preventing private and confidential data from leaking through 'why'-questions, and efficiently maintaining the data necessary to answer such questions. We also report some early experience with a prototype system we have been working on.

2 Self-explaining networks

To explain the concept of self-explaining networks in more detail, we begin by discussing a simple example scenario, in which an adversary (evil Eve) compromises part of the Internet's interdomain routing system and diverts traffic in an attempt to eavesdrop on another user (Alice). We focus on interdomain routing here because it is known to be plagued with a variety of well-studied problems, ranging from benign equipment failures to deliberate attacks [18]; however, we expect our idea to be applicable to other networks, and perhaps even distributed systems in general.

Consider the scenario in Figure 1, in which Alice is using a service at `foo.com`. In the original system (left), network A's policy does not allow cross traffic, so Alice's only viable route is r_1 through networks B and C. If Eve is able to compromise A's router, she can change this policy (right) and thus cause Alice's router to switch to the new, shorter route r_2 through network A, thus enabling Eve to eavesdrop on Alice's traffic. However, note that Alice's system administrator (Bob) cannot observe the cause of this change, only its effect. Bob may notice the new, unfamiliar route, but he cannot easily determine *why* it appeared.

Moreover, Eve can thwart an investigation by making the compromised router lie to Bob. For example, she could make it appear as if the route through A had always existed, or that the cause of the change was an event in another network. This may send Bob in the wrong direction or cause him to abandon the investigation because everything appears to be fine. To enable a reliable, timely response to such attacks, we need a *secure* tracking system that leaves even the smartest attacker no chance to escape detection.

2.1 Provenance and the provenance graph

Our goal in this case is to enable Bob to ask *why* the new route appeared, in such a way that Eve cannot lie without giving herself away. In the absence of adversaries, this can be done by using *data provenance* [3]. A data provenance system tracks all dependencies between data in the system – for example, that network B got the route to `foo.com` from network C, and that it exported it to Alice's network after applying some transformations to it. Conceptually, these dependencies form a global *provenance graph*, whose nodes represent data and whose edges represent processing steps.

Many 'why'-questions (and, more generally, questions about causes and effects) can be answered by subgraphs of the provenance graph. For example, to determine the causes of a datum d, we can traverse the graph upwards from d until we arrive at a set of base nodes, such as local inputs, which have no further dependencies. Similarly, we can determine the effects of d by traversing the graph downwards to a set of leaf nodes. As an example, Figure 2 shows the subgraph that explains the switch from route r_1 to r_2 at Alice's network as being caused by Eve's policy change on router r_2.

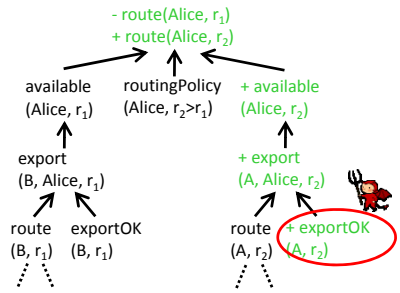

Figure 2: Provenance tree explaining a routing change ("+" and "–" indicate value insertion and deletion).

2.2 Ideal solution

An ideal self-explaining network would be able to return such responses even when it is under attack. In particular, the responses would be complete and correct, i.e., contain the whole truth, and nothing but the truth. However, recall that we have assumed that an adversary may have *complete control* over an *arbitrary subset* of the nodes in the system, which enables her, for example, to refuse to record provenance information, lie to the querier, suppress information on the nodes she controls, or even forge information to make honest nodes appear faulty. The only thing we assume she cannot do is invert hash functions or forge cryptographic signatures.

It seems clear that, under these conditions, building the ideal system is impossible. The provenance information must necessarily be stored somewhere, and, in the absence of trusted components, there is always a possibility that the adversary has compromised exactly the nodes that are storing the telltale information.

2.3 Practical approximation

While the ideal system seems infeasible, we can at least build something very close to it, based on two key insights. First, we can restrict the guarantees to *observable* behavior, i.e., actions by the adversary that directly or indirectly affect at least one correct node. This limitation is fundamental [6]. Second, we can weaken the requirements a bit and accept that nodes may lie temporarily, *as long as lies are eventually detected and identified as such*. In other words, Bob might briefly see a plausible but incorrect explanation of the bad route, but soon afterward the system reports the compromised router and flags the affected parts of the explanation. As long as the detection period can be kept short (on the

order of network round-trip times), this does not seem unduly restrictive, and it is sufficient to arrive at a practical solution.

3 Requirements

We now turn to the question how to build such a practical self-explaining network, which will need at least the following four components:

- A mechanism for tracking the provenance of the state in the network and for constructing the provenance graph;
- A storage system for securely and efficiently storing the provenance;
- A query language for asking 'why'-questions; and
- A query processor for answering 'why'-questions in such a way that inaccurate or missing information from compromised nodes can be detected.

In designing these components, we can build on a body of prior work in both the systems and the database community, e.g., on tracking provenance [17, 25], tamper-evident logging and fault detection [7], and query processing for provenance data [9]. Also, there is existing work on multi-host forensics [2, 12]. However, existing solutions typically require trust in at least some components of the system, such as the OS, the hypervisor, or some special hardware; in fact, many existing provenance systems are designed for scenarios where all nodes are correct. Since we would like self-explaining networks to work "when all else fails", we want to avoid such trust assumptions to the extent possible.

Strawman: PeerReview + ExSPAN. At first glance, there seems to be a simple way to build a self-explaining network, namely combining a provenance system with a tamper-evident log and a fault detector. However, when dealing with a malicious adversary, such ad-hoc solutions rarely work. Consider, for example, a strawman solution that consists of two state-of-the-art solutions, PeerReview [7] and ExSPAN [25] . PeerReview can detect a large set of faults and misbehaviors, while ExSPAN can extract and query provenance data from distributed applications written using *Network Datalog* [13]. It would appear, therefore, that a combination of the two could detect faults *and* answer provenance queries. However, this strawman approach is in fact insufficient. The combination could not give any meaningful guarantees on query responses once nodes have been compromised. This is because although PeerReview would correctly detect such nodes, the compromised nodes could corrupt the provenance information on other nodes before they can be evicted, causing future query responses to be inaccurate or even correct nodes to appear faulty. We will outline these challenges and more in the next section.

4 Towards a practical system

In order for self-explaining networks to have strong and provable properties, we need to design a provenance model and a query processor *specifically for the adversarial setting*. In addition, there are numerous other challenges that have to be solved, such as OS support for extracting provenance, storing the provenance securely, or preventing private and confidential data from being exposed through the provenance records.

4.1 Provenance model

The classical notion of provenance focuses on currently extant state in a quiescent system. This is not sufficient for an adversarial setting, for at least three reasons.

First, by the time a symptom is noticed, the adversary may already have covered his tracks by deleting the original cause from the network. Thus, we need a way to query the provenance of past states. Also, if provenance is only guaranteed to work when the network is quiescent, the adversary can prevent forensic queries, e.g., by causing oscillations. To capture past and transient states, we need to extend the provenance model with a *temporal* dimension. This can be done by attaching a time interval to indicate when state is (or was) valid, and by capturing information flow through messages.

Second, merely explaining extant state is not enough. Sometimes the event of interest is a *change* from one state to the other, or the appearance or disappearance of state. To produce meaningful explanations of state changes, we need to capture dependencies between state changes as well.

Finally, since there is no central trusted component that could store the entire provenance graph, we need a way to partition the graph in such a way that each node can store a piece of it. The natural way to do this is to have each node store the provenance of the state it maintains locally. However, classical provenance graphs contain certain vertices that pertain to more than one node; for example, a message transmission pertains to both the sender and the receiver. If one of them were allowed to keep the entire vertex (as is done in ExSPAN's *network provenance* [25] model), an adversary that compromises this node could manipulate the vertex and potentially make the other (correct) node appear faulty. To prevent this, we must split and interconnect such vertices so that the graph can be partitioned cleanly.

4.2 Extracting provenance

A challenge faced by any provenance system is to extract the provenance of data from the target application. Depending on how the network components are implemented, this can be done in at least two ways. If the component is being designed from scratch or the source code is available, the developers can add explicitly *declare* the dependencies between network states by adding provenance annotations [17]. For this approach to work, the developers must have a good understanding of the source code, since they are responsible for the correctness of the declaration. Alternatively, it is possible to *infer* coarse-grained dependency relationships from a component's inputs and outputs. In this approach, components can be treated as black boxes (for reasons such as business secrets, missing source code, or lack of extensive understanding of the system). Support from operating systems is desirable: an OS-level wrapper around the target applications can actively capture inter-node (or inter-processes) communications and other system calls, providing us with a convenient interface to extract I/O for inference.

4.3 Storing provenance

To detect when nodes lie about provenance, we must store the provenance information in a suitable data structure. Here, we can build on previous work on tamper-evident logs [7]. Briefly, this involves signing and acknowledging all messages, as well as exchanging information about each message with some other nodes to detect inconsistencies. This does not actually *prevent* tampering, but it ensures that correct nodes can *detect* when a compromised node tampers with its log, which is consistent with our goal from Section 2.

A serious concern for self-explaining networks is efficiency. Many state-of-the-art provenance systems [5, 17, 25] actively maintain provenance for each piece of extant state. If this were done for all past states as well, the overhead would be enormous. Our key insight is that we can expect queries to be relatively rare (they are needed only when an attack is suspected), so we can trade some query performance for better storage efficiency. We can do this by storing not the entire provenance graph, but only enough information to *securely reconstruct* the parts of the graph that are needed to answer a given query. For example, if the network implementation is deterministic, it is sufficient to have each node store all inputs; the rest of the graph can be reconstructed through deterministic replay. If checkpoints are recorded periodically, replay can start at the nearest checkpoint rather than at the beginning of the log.

4.4 Querying provenance

Querying is the heart and soul of a self-explaining network. To be maximally useful, the system should support a variety of provenance queries, and, of course, it should be possible to tell from the responses whether a compromised node has lied.

Graph-based query language. The most obvious questions to ask a self-explaining network are 1) about the causes of a network state s, and 2) about the effects of s. The algorithm for answering such questions could easily be hard-coded. However, we expect that there are many other questions that could profitably be asked – and be answered using information the system already maintains – such as the question whether a particular invariant that spans multiple nodes was ever violated. Given that provenance essentially forms a graph, we conjecture that graph-based languages such as ProQL [9] are promising candidates for formulating arbitrary queries and transformations over provenance data.

Secure recursive querying. If the optimization from Section 4.3 is used and the network records only enough data to reconstruct the provenance graph when necessary, another challenge arises, since the replay-based reconstruction itself might be compromised if it is carried out at a remote node. To avoid this, we can always ship the logs to the querying node and perform the reconstruction there. This is possible because data integrity can in principle be verified through the tamper-evident log; however, doing so efficiently requires additional support because the original data structure can only verify the entire graph, not individual subgraphs. Thus, the reconstruction process starts from the network state specified in the query and recursively traverses the provenance graph, requesting subgraphs from nodes as necessary, until base or leaf vertices are reached.

4.5 Privacy and access control

In some instances, it may be necessary to treat provenance data as confidential – particularly if the self-explaining network has more than one administrative domain (e.g. ASes in the inter-domain routing). In this case, each domain may want to restrict what provenance information can be seen by nodes in other domains. Thus, it should be possible for provenance information to be hidden (encrypted) based, e.g., on roles or security levels. A naïve implementation of provenance may result in information leakage by exposing sensitive information to the recipients of the provenance data, some of which may be unauthorized to access the leaked data. We conjecture that one can utilize information hiding techniques [15] to support fine-grained confidentiality controls, by partially encrypting provenance in a key-

based tree-structured architecture. Another promising approach would be to support a multi-level provenance model [17, 19] and to evaluate queries only within certain levels.

4.6 Overhead

Two main sources of the overhead incurred by provenance systems come from the maintenance and querying of provenance. Noticing the relative rareness of provenance queries, we bias the priority to minimizing the maintenance overhead (discussed in Section 4.3). We expect the maintenance overhead (mainly on storage) is affordable given the vast storage units available at a reasonable price. Our preliminary experience echos this expectation (see Section 5).

On the other hand, provenance querying, incurs high communication overhead due to the need of shipping logs and snapshots to the querying node. While it is not impractical, considering the rareness of provenance querying, it does deserve further optimizations. For instance, the nodes involved in a query may only ship the logs containing the events that may contribute to the final result.

5 Status and ongoing work

We are currently working on a prototype framework for self-explaining networks that uses ExSPAN [25], with the extended provenance model described in Section 4.1, for maintaining and tracking network provenance, RapidNet [21] for distributed execution of ExSPAN queries, and PeerReview [7] for distributed tamper-evident logging. An additional *proxy server* is utilized to intercept incoming and outgoing messages of each node and feed them into the system. In addition, we are currently enhancing our framework by addressing each of the challenges described in Section 4.

As a proof of concept, we have applied this system to two applications: Quagga [20], an open-source implementation of BGP, and a declarative implementation of the the Chord distributed hash table [14]. Our initial results indicate that the overhead will be low enough to be practical. For example, a moderately sized Internet network would need to record only 492 GB provenance data per year, which can easily fit onto a commodity hard disk. The communication overhead is a constant factor over the original traffic and does not affect the scalability of the protocol. The aggregate traffic for answering a query is less than 2 MB on average, and the result is returned within 18 seconds.

6 Acknowledgments

This work is supported by NSF grants CNS-1040672, IIS-0812270, and DARPA award N66001-11-C-4020.

References

[1] M. Basrai and P. M. Chen. Cooperative ReVirt: adapting message logging for intrusion analysis. Technical Report University of Michigan CSE-TR-504-04, Nov 2004.

[2] M. Basrai and P. M. Chen. Cooperative ReVirt: Adaptive message logging for intrusion analysis. Technical Report CSE-TR-504-04, University of Michigan, 2004.

[3] P. Buneman, S. Khanna, and W. C. Tan. Why and Where: A Characterization of Data Provenance. In *Proc. ICDT*, 2001.

[4] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. In *Proc. SOSP*, Oct 2007.

[5] T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. ORCHESTRA: Facilitating Collaborative Data Sharing. In *Proc. ACM SIGMOD*, 2007.

[6] A. Haeberlen and P. Kuznetsov. The Fault Detection Problem. In *Proc. 13th Intl. Conf. on Principles of Distrib. Systems (OPODIS)*, Dec. 2009.

[7] A. Haeberlen, P. Kuznetsov, and P. Druschel. PeerReview: Practical Accountability for Distributed Systems. In *SOSP*, Oct 2007.

[8] R. Hasan, R. Sion, and M. Winslett. Preventing history forgery with secure provenance. *ACM Trans. Storage*, 5(4):1–43, 2009.

[9] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *Proc. ACM SIGMOD*, 2010.

[10] B. Kauer. OSLO: Improving the Security of Trusted Computing. In *Proc. USENIX Security Symposium*, 2007.

[11] S. T. King and P. M. Chen. Backtracking intrusions. *ACM Trans. Comput. Syst.*, 23(1):51–76, 2005.

[12] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen. Enriching intrusion alerts through multi-host causality. In *Proc. NDSS*, 2005.

[13] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative Networking. *CACM*, 52(11):87–95, 2009.

[14] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing Declarative Overlays. In *Proc. SOSP*, 2005.

[15] G. Miklau and D. Suciu. Controlling access to published data using cryptography. In *Proc. VLDB*, 2003.

[16] D. Moore, C. Shannon, and k. claffy. Code-Red: A Case Study on the Spread and Victims of an Internet Worm. In *Proc. ACM IMW*, 2002.

[17] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *Proc. USENIX ATC*, 2006.

[18] O. Nordstroem and C. Dovrolis. Beware of BGP attacks. *ACM Computer Communications Review (CCR)*, Apr 2004.

[19] C. Olston and A. D. Sarma. Ibis: A provenance manager for multi-layer systems. In *Proc. CIDR*, 2011.

[20] Quagga Routing Suite. `http://www.quagga.net/`.

[21] RapidNet. 2010. `http://netdb.cis.upenn.edu/rapidnet/`.

[22] K. Shanmugasundaram, N. Memon, A. Savant, and H. Bronnimann. ForNet: A distributed forensics network. In *Proc. MMM-ACNS*, 2003.

[23] R. Wojtczuk. Subverting the Xen hypervisor. Black Hat conference, Aug. 2008.

[24] Y. Xie, V. Sekar, M. Reiter, and H. Zhang. Forensic analysis for epidemic attacks in federated networks. In *Proc. ICNP*, 2006.

[25] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient Querying and Maintenance of Network Provenance at Internet-Scale. In *Proc. ACM SIGMOD*, 2010.