Executive Summary

02/11/2010

Andrew Arminio
Christopher Austin
James McCauley

Introduction

While home and small office/home office networks have provide any number of benefits -- internet access, media sharing, shared printers, etc. -- they seldom live up to their potential. Many such networks are subject to security problems, some of which allow attackers access, and some which negatively impact usability by providing *too much* or *the wrong* security. These networks also often simply do not provide for as good of a user experience as possible, for example because bandwidth provisioning in such networks is simple-minded.

While our client, Nicira Networks, focuses on large networks such as enterprise and datacenter networks, the technologies they use and develop could serve to iron out the rough spots of home/SOHO networks, and allow for whole new classes of home network capabilities. It is the task of Team NOX and the NOX At Home project to make this a reality.

Problem Statement

In 1981, the seminal paper "End-to-End Arguments in System Design" by Saltzer et al. famously articulated a design principle that had already been key to the design of communication networks for quite some time. The case was presented that many network features needed to extend all the way from one endpoint to the other -- from a sending application to the receiving application -- and that given that requirement, the correct placement of those features was in the endpoints and not in the underlying communication system. In essence, this espouses a model that is often summed up as "dumb network; smart hosts." There is no debate that the arguments of the paper are valid and that the principle is a solid one. However, not all design elements are best served by this principle and this principle alone. Although Saltzer and his coauthors address this in their paper, the

principle itself and the "dumb network; smart hosts" model essentially became network design dogma.

However, a dumb network is not without its problems, and there are bona fide reasons for wishing to build smarter networks -- to put functionality into the network itself. Corporate networks, for example, are often subject to strict reliability and security constraints which do not fit well with the IP's "best effort" approach. To resolve this, there has been considerable effort to make corporate networks more manageable, essentially by "bolting on" features (firewalls, VLAN control, audit logs, etc.). While these may provide tools for management, one might summarize these developments as "working harder, not smarter."

One reason for the lack of smarter networks is simply that adding any real "brains" to a network has posed at least one question that has been difficult to answer: where do these brains *go*? There would seem to be two basic approaches -- a distributed one or a centralized one -- and neither of these has been incredibly appealing. The distributed approach is difficult at best simply due to the inherent difficulty of distributed solutions for the complex and variable nature of network management. In the case of internetworks, this approach would often be entirely impossible because administrative boundaries between the owners of different networks. On the other hand, centralized approaches have long been decried in the networking world. There are (justified) concerns about having a single point of failure and about scalability.

Challenges aside, there were real problems to be solved for which adding functionality to the network seemed to be key to the solution, and this spurred a string of research at Stanford University. The first of this research was SANE; a clean-slate approach that strived to make make enterprise networks more secure and to make that security easier to manage. This was followed by Ethane, a major refinement that, significantly, was not a clean-slate approach; it was incrementally deployable in existing networks. Finally, this brings us to NOX and OpenFlow. NOX and OpenFlow correspond to the two major components of Ethane, but where Ethane was academic and aimed at one specific goal (security in enterprise level networks), NOX and OpenFlow are real solutions that have been generalized for a multitude of purposes, including ones that are, as yet, undefined. NOX and OpenFlow work hand in hand with the aim of separating low-level packet forwarding from high-level decision making. Said another way, they make the network itself programmable, which enables smart networks.

OpenFlow is the part that manages the low-level packet forwarding, and is an open standard governed by the OpenFlow Switch Consortium. This standard defines an interface

to control an abstract *flow table* in a network switch. Through manipulation of this flow table, one can inspect network traffic, control how packets are forwarded (or not forwarded!), and alter the traffic in other ways (such has packet header rewriting and QoS management). Hardware routers implementing OpenFlow exist today (indeed, such support is increasing), and there are also software implementations that transform a Linux machine into an OpenFlow switch. OpenFlow, which received the Best Demo award at SIGCOMM 2008, reached version 1.0 at the end of 2009. This release was a major milestone, representing the first version of the standard intended for general availability in commercial products.

By itself, an OpenFlow switch does nothing -- it requires a controller to make high-level decisions. By far, the most predominant of these is NOX. NOX is an open source "operating system" for networks that provides a centralized programming interface to the entire network via its ability to control OpenFlow switches. It should be noted that while NOX is *logically* centralized, it need not actually be a completely centralized system. The actual machines running NOX can be replicated, implement failover, be load-balanced, or utilize other modern datacenter designs aimed at improving reliability and scalability (which did not, for the large part, exist twenty years ago). Further enabling scalability is that much of the "grunt work" is handled by the switches. While performance will obviously be impacted by the specific functionality implemented on top of NOX, Ethane (using the same basic approach and architecture as NOX) was able to manage over 5,000 hosts from a controller running on a single commodity PC.

Our client, Nicira Networks, is the primary developer of NOX, as well as the primary developer of Open vSwitch -- a virtual network switch that, among other things, implements OpenFlow. Nicira's focus is and continues to be on on applying NOX and OpenFlow technologies to campus, corporate, enterprise, datacenter, and virtual networks. However, they also see that these technologies may be of benefit to home and small office/home office networks. While these are not the markets they are focused on (or they would surely develop a solution internally!), they do have number of reasons to wish to see progress in this area, and that is the goal of NOX At Home. Specifically, they have a number of high-level goals: - Create a system of NOX applications suitable for putting in to the firmware of a replacement for the ubiquitous "WiFi router." - Explore the implications of NOX and OpenFlow technologies in a new arena. - Raise awareness and increase adoption of the technologies among tech-savvy users. - Improve the state of home/SOHO networking and the internet at large.

This last goal is perhaps the easiest with which to identify. Almost everyone has either directly experienced the shortcomings of the "state of the art" in home/SOHO networking or is familiar with those shortcomings: - Home network security is difficult for users, and the failure to properly configure various components leads to suboptimal results. The most obvious of these, of course, is susceptibility to attackers, and there should be no need to detail the very real threat of such attackers even for ordinary home users (estimates of botnet penetration being what they are -- numerous estimates are up to 20% of all hosts on the internet). However, they are more subtle problems here as well. For example, WiFi security is difficult for many users to configure, and many people simply leave it disabled. Users capable of configuring it may leave it off anyway or purposely use a weak password for practical reasons -- it's too annoying to use it properly when you have guests that you want to allow to use your wireless. In any case, this is all or nothing: users either have access or they do not. To confront the security problems presented by wireless, it is often advisable that each machine on a LAN have its own firewall and be running a very limited set of services, but this limits functionality that would actually be convenient such as filesharing. The result? Many users routinely email themselves documents so that they can access them on a machine that's ten feet away. - The only global access control is extremely course-grained. While some home routers have port filters, time-based and web site restrictions, and other similar features, these tools are always across low level names (such as IP addresses) and not the high level names that would make them truly useful. - There is no global system for the management of bandwidth. Home networks generally attempt to be *fair* about the allocation of bandwidth, but this does not account for the fact that not all traffic is created equally. For example, long downloads should not have the same priority as streaming video or telephony or even interactive web browsing. - Home/SOHO networks lack visibility. Is your network being used as a spam relay? Are kids using the internet when they are not allowed to? Are employees using peer to peer filesharing at work? Are neighbors stealing your wireless? These questions are all relatively hard to answer with current home networks.

In short, our project boils down to a single statement: Let's use NOX and OpenFlow to make home networks better.

Product/Solution Statement

There is more potential for NOX and OpenFlow in the home network than we could possibly develop in a semester. Our true goal is not to deliver the end-all-be-all of home

networking. Our goal is simply to get the process started -- to provide the proper "jumping off point" from which future home/SOHO networking tools can be built. This means we must cater to two separate audiences: the technical audience of people wishing to do network research or develop specific network "applications" for the home, and nontechnical consumers who simply wish to take advantage of these tools.

Our solution can actually take some hints from the three principles that drove the development of Ethane:

The network should be governed by policies declared over high-level names. The average home "WiFi router" does have some ability to set policies -- port forwarding, port filtering, sometimes even time-based access control -- but these policies are universally set over IP addresses which mean nothing to average end-users. We need provide the ability to set policies on the names of users and groups of users ("Perry", "The Kids"), recognizable machine names ("John's laptop", "Susan's Dell", "Canon Printer"), and useful network names ("WiFi", "The Internet", "House Network").

Network routing should be policy-aware. In the context of Ethane, this meant the selection of specific network paths based on policy. For NOX At Home, the situation is much simpler (the paths over which we have control are very few), but the basic idea still holds: *policy* decides what packets are sent where (and which are dropped altogether).

The network should track bindings between network traffic and the high-level entities that are its source and destination. The final principle behind Ethane was "The network should enforce a strong binding between a packet and its origin." This principle makes sense for enterprise networks where security is of primary concern -- the ability to tie network traffic to specific users and machines closes many avenues of attack for network intruders. However, the home network differs from the enterprise network here: in an enterprise network, globally managed users with strong and explicit authentication are the norm, and this is not the case for the home network. Given this, it is suitable to somewhat relax this final principle. We also wish to slightly expand on this statement, because not only the *origin* (or source) of the traffic but also the *destination* may factor in to to policy selection. What we are really saying here is that we need to be able to associate traffic with high-level names in order to correctly make policy decisions for that traffic.

Finally, our solution has some "user experience" aspects that were not part of Ethane at all:

An ordinary consumer end-user should be able to operate the system with a minimal investment of time. While we hope that the use of high-level names will be a great step in

this direction, this also very clearly calls for a graphical user interface that leverages the investment that users have already made in operating personal computers.

Users should not find the system to be a burden to use. Our goal is to enable users to get *more power*, *more usability*, and *more security* from their home network. Our goal is *not* to to make their network *more annoying* or *waste more of their time*. This principle mandates some compromises. For example, while we may be able to get more security by requiring explicit logins, we do not feel that this would be acceptable to many users: despite any improvements we may be able to offer, users will feel that our system is annoying and is getting in their way. We must aim to make whatever improvements we can while impacting user experience as little as possible.

These high level goals motivate the development of a framework that enables the development of tools to improve home/SOHO networking. However, a framework based purely on high level goals is both difficult to design and difficult to evaluate. To this end, we have identified and will build some prototypical home/SOHO network tools. The successful completion of these tools will then not only serve to provide the capabilities of the tools themselves, but to validate the design and implementation of the framework and the achievement of the high level goals. The specific features we have identified are:

- Identify machines by MAC address
- Identify users using passive methods (snooping on Instant Messaging and website account logins)
- Identify users actively -- through an explicit web login/logout page
- Maintain and display a view of the network -- which machines are online, and which users are using them
- Allow certain users/hosts to be disallowed LAN access and given only Internet access
- Allow certain users/hosts to be allowed only HTTP and have other traffic filtered
- Give users feedback on events that have occurred (access to the LAN denied, etc.)

While these features are certainly not the most compelling that this platform should have to offer, we feel that their implementation will properly guide the development of the platform and the implementations of further components. Specifically, we have identified the following major components (or classes of components) that will be needed to support these features:

- **Evidence Sources**. Evidence Sources provide a mapping between network traffic and our high-level names -- users and machines. An "active" source would be a explicit login

over a "captive portal" type web page, but we believe the more important sources will be passive and based on the interpretation of network traffic. For example, an evidence source could be written that monitors for network traffic for instant message conversations and knows how to associate one of the network's users with a particular instant messaging account.

- **Network View**. The Network View maintains a view of the network state -- which users are on the network, which machines are on the network, which users are on which machines, etc. This is done primarily through the evaluation of evidence given by Evidence Sources. We suspect that evidence will seldom be as complete, authoritative, and non-conflicting as we would like, so evidence must be combined and arrive at a "best guess" of the truth. This will probably be done using existing research done on this subject, such as that done by Bayes, Dempster, Shafer, and Hooper.
- **Policies**. Policies actually affect network traffic. In some cases, these may be relatively simple, such as "never allow traffic to example.com". In many cases, however, we believe that effective policies will involve high-level names, such as "disallow VNC access to Tom's Linux Box to any user except Tom" or "don't allow Guests access to anything on the LAN except for Printer2". For this reason, many policies will query the Network View to gain insight into these high level names.
- **User Interface**. We wish to be able to allow for easy user/machine management, as well as the selection of policies. We also wish for future components to easily integrate with this same user interface. A web based user interface is ideal for our system, as it will allow users to access the interface from any machine on the network and requires nothing to be installed, which plays nicely with our "minimal investment of time" and "not a burden" principles.

The user interface's "web app" portion will be built using the qooxdoo framework, though we have not yet identified solution to the user interface back-end. Evidence Sources, the Network View, and Policies are all individual NOX applications. Evidence Sources and Policies will generally provide their own user interfaces (seamlessly integrated into our web app).

Requirements and Functional Spec

The motivations for and principles driving our project translate to a number of specific requirements and an associated functional specification.

Allow administrators to configure policies on their network using high level names. Administrators must be able to configure policies for users and/or machines on their network as well as groups thereof. This includes loosely defined classes of users such as anonymous users or invited guests (e.g. an administrator may have policies set up for any friend who is using the network temporarily), and policys for unknown users. To enable this functionality, we need our system allow our users to:

- Add/remove users or machines
- Create groups of users and/or machines
- Add/remove users or machines to/from groups
- Configure evidence sources for individual users or machines
- Configure policy applications for individual or groups of users or machines.
- Configure policy applications for unknown users and machines.

Identify users and machines on a network using active and passive mechanisms. Active and passive methods of user and machine identification must be able to provide the system some degree of confidence that a specific user or machine is active on the network at a given time. New evidence applications must have a way to interact with the system. Specifically, the software must:

- Intercept any network traffic with a specific criteria.
- Look at intercepted network traffic for evidence that it belongs to a specific user or machine.
- Accept direct user authentication through a *captive portal* type interface.
- Identify where (IP, MAC, previously identified machine) a user is active.
- Alert the rest of the system that a user or machine is known to be or suspected to be active. In cases where a user or machine is only *suspected* to be active, information about the likelihood must be provided.
- Identify new evidence applications and incorporate them into the system.

Maintain a view of the current network state. The system should be able to aggregate the messages from various evidence sources and make a decision about their meaning. These decisions should be recorded as a coherent and comprehensive view of what users and machines are active on the network at any given time. To support this, the software must provide the following functionality:

- Aggregate and store messages from evidence applications.
- Analyze such evidence messages and make a decision on where and if a user is active on the network.
- Store decisions about active users and machines on the network as the network state.

- Provide network state information to policy applications.
- Provide network state information to administrators using a web interface.

Integrate arbitrary network policies. The system needs to provide a means for independently developed network policy applications to run. They should have the ability to access network state decisions from the network view and the ability to be configured from the main web interface. Our system must provide the following functionality to support such policy applications:

- Identify new policy applications and incorporate them into the system.
- Have a means for such applications to retrieving information from the network view.
- Have a means for such applications to know what users/machines/groups are set up on the system.

Constraints and Feasibility Issues

As the NOX At Home project is to be open source software, all outside components used (libraries, media, etc.) must have compatible licenses.

The framework must be capable of being run on its (rather limited) target hardware platform: PC Engines alix single board computers (in particular, the alix2c3 and alix2d3).

One large constraint is to do with testing. While we can run the system on our own home networks, home and SOHO network size, composition, and usage vary considerably from network to network. It is unlikely that we can test under even a truly representative fraction of the variations.

Perhaps the largest feasibility issue is with passive user identification. As far as we know, our system is novel in its approach here. As such, the very underlying ideas are unproven, and there is no prior work to guide our design or implementation. Using it in this project is, in fact, a relatively risky attempt to skip a lengthy research phase and jump straight into a production system.

Project Execution Plan

In order to achieve our goals this semester, we plan on following a modified Lean development process. The process will consist of five milestones during which certain functionality are expected to be completed. At the beginning of each milestone, functionality will be divided into chunks and written on cards. Cards will then be assigned to team members. During each milestone the pieces of functionality will be tracked through

development phases through the use of a kanban or "visual board". The kanban is broken up into sections correlated with development phases, and the cards representing functionality will be placed and moved within the kanban as appropriate. At the end of each milestone, the development process will be evaluated for efficiency. The process will then be modified per that analysis.

During the initial milestone run, each card will pass through the following three development phases:

Research - In this phase most of the research needed to work on a card is completed, including any major technical hurdles. If, during the course of the research, it is discovered that the original card can not be completed as written it may be broken up, rewritten, or put off until the team can make a decision on it. If the card is sufficiently difficult, a stripped down version of it should be completed, as a proof of concept, before passing on to the development phase.

Development - During this phase the functionality specified on the card should be built. If unexpected difficulty is encountered during this phase the card may be returned to the research phase.

Test - During this phase a developer other than the one who worked on it during the development phase should review the card and make sure it performs as specified. Cards that fail this phase may be passed back to the development phase.

Milestones

Our milestones have been selected as "sequence points". We feel that the items in a particular milestone would be best served if the items in the previous milestone are complete. As we are using a largely agile approach, "complete" is a somewhat relative term. At any point, we may wish to (or be forced to) revisit any item. However, we do wish for the items from previous milestones to be *working*.

The dates are not firm deadlines and are mostly meant to ensure that we keep scheduling on track.

Milestone 1 (March 4th)

- Evidence Apps
  - Machine MAC Address
  - User Email or IM
- Basic Network View with Limited UI

Milestone 2 (March 18th)

- Captive Portal Evidence Source
- Web App
  - Login and Session Management
  - Basic Network Browser UI
  - Basic Inbox

Milestone 3 (April 8th)

- Simple Policies
  - Internet Traffic Only
  - HTTP Only
- Web App
  - UIs for Existing Evidence Apps
  - UIs for User/Machine/Group Management

Milestone 4 (April 22nd)

- UIs for Policy Configuration
- "Other Functionality" UI Framework
- Generic Evidence UIs

Milestone 5 / Completion (May 6th)

- User Testing (will not follow kanban development process)
- UI Review

Further Reading

End-to-end arguments in system design. Saltzer, J. H., Reed, D. P., and Clark, D. D. 1984. ACM Trans. Comput. Syst. 2, 4 (Nov. 1984), 277-288. DOI=http://doi.acm.org/10.1145/357401.357402

SANE: A Protection Architecture for Enterprise Networks Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman Dan Boneh, Nick McKeown, Scott Shenker {casado,talg,mfreed,dabo,nickm}@cs.stanford.edu aditya@cs.cmu.edu, shenker@icsi.berkeley.edu

Rethinking Enterprise Network Control Martin Casado, Member, IEEE, Michael J. Freedman, Member, IEEE, Justin Pettit, Member, IEEE, Jianying Luo, Member, IEEE, Natasha Gude, Member, IEEE, Nick McKeown, Fellow, IEEE, and Scott Shenker, Fellow, IEEE,

Rethinking the design of the Internet: The end to end arguments vs. the brave new world Marjory S. Blumenthal, Computer Science & Telecommunications Bd., mblument@nas.edu David D. Clark, M.I.T. Lab for Computer Science, ddc@lcs.mit.edu