Delegating Network Security with More Information

Jad Naous
Stanford University
California, USA
jnaous@stanford.edu

Ryan Stutsman
Stanford University
California, USA

David Mazières
Stanford University
California, USA

Nick McKeown
Stanford University
California, USA
nickm@stanford.edu

Nickolai Zeldovich
MIT CSAIL
Massachusetts, USA
nickolai@csail.mit.edu

ABSTRACT

Network security is gravitating towards more centralized control. Strong centralization places a heavy burden on the administrator who has to manage complex security policies and be able to adapt to users' requests. To be able to cope, the administrator needs to delegate some control back to end-hosts and users, a capability that is missing in today's networks. Delegation makes administrators less of a bottleneck when policy needs to be modified and allows network administration to follow organizational lines. To enable delegation, we propose `ident++`—a simple protocol to request additional information from end-hosts and networks on the path of a flow. `ident++` allows users and end-hosts to participate in network security enforcement by providing information that the administrator might not have or rules to be enforced on their behalf. In this paper we describe `ident++` and how it provides delegation and enables flexible and powerful policies.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations; C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Management, Security

Keywords

ident, firewall, network security, policy, management

1. INTRODUCTION

While network security policy is usually decided by a single authority, the network administrator, that administrator has usually had to configure myriad security devices, firewalls, and end-hosts. Gradually, the configuration of the network has become more centralized, enabling an administrator to configure a consistent security policy at a single location and have it enforced across various devices. Recent proposals take centralization even further, proposing that almost all network features be pulled out of the datapath into a central controller [6, 5, 8, 10, 1], giving the administrator direct control over routing, mobility, and access control. We expect this trend to continue.

While there are many advantages to centralizing the control of enterprise networks, such centralization places a heavy burden on the administrator. She needs to manage an increasingly complex set of rules, respond to user requests, and handle network upgrades and extensions. The administrator becomes more prone to errors as security policies become large and complex, and she becomes the bottleneck when the need to alter policy arises. For instance, users may need to configure their own machines over the weekend (when the administrator may not be available). Departments may need to control their own policy (such as who may access their servers) at a fine enough granularity that the cost of having an administrator with different priorities in the loop is intolerable.

And even when administrators are able to handle the full network complexity, they often do not have the precise information they need to make a correct security decision. So the administrators are forced to write coarse network security policies that cripple legitimate use of the network. For example, the administrator may wish to deny Skype access to an important webserver but is unable to because Skype and Web traffic both use destination port 80. This information is usually only available at the end-hosts, and is often unavailable when making security decisions.

Limited administrator resources and the over-broad security policies can harm productivity significantly and frustrate users, enticing them to bypass security protocols. However, these problems can be overcome by delegation. An administrator in a large organization might want to delegate some control to a department or site administrator, or might want to delegate specific aspects of network policy to an end-user. And while delegation was desired in previous network architectures that were centrally managed, only more recent architectures with strong central control make it possible to *delegate* control (i.e. provide restricted privileges for others to control parts of the network), *log and audit* the delegates' actions, and *revoke* the delegation if needed. The administrator can finely tune the degree of delegation to balance security concerns with productivity and morale needs.

WREN'09, August 21, 2009, Barcelona, Spain.

We propose ident++, a protocol designed to help network administrators delegate aspects of network security policy to other decision-makers—end-hosts, users, applications, application developers, or even trusted third-parties. ident++ is simple; it is inspired by the Identification Protocol [14], but is richer and more flexible. An ident++ daemon running on senders and receivers responds to queries from ident++-enabled decision-makers about flows. The response to a query is a list of key-value pairs that can be used to provide information upon which the decision-maker can make its decision. ident++-enabled decision-makers on the path can also answer queries on behalf of end-hosts, or can modify responses to add additional information.

Delegation in ident++ is two-fold: it involves the end-hosts and users in classifying traffic and it allows them to specify rules to be enforced in the network using the key-value pairs. These pairs are mostly free-form and ident++ does not constrain the types that can be used. ident++ specifies a few—such as user, application name, hash of the executable, and user-defined allow/drop rules—and it allows administrators, users, and application developers to specify their own.

ident++ allows policies to be described in terms of principals rather than incidental flow properties, a position already advocated in [5]. In fact, any of the keys returned in the ident++ response packet can be used as a principal giving greater flexibility than previous approaches. This allows policies to be written free of network-level properties—such as IP addresses and TCP port numbers—with which the security policy is not concerned. For example, by blocking port 25, the policy is trying to block SMTP traffic to recipients who don't want it, but the collateral damage is that users who wish to use SMTP authentication cannot relay mail through their own servers.

Our contributions are the following:

- A protocol to query senders, potential receivers, and networks along the path for user-definable additional information on a flow.
- Extending OpenBSD's PF [2] language to enable controlled delegation and flexible rules.
- A system design we intend to implement on OpenFlow [12].

In §2, we provide an overview of how ident++ works. Then we describe the design of ident++ in more details in an OpenFlow network in §3, and present some policies and applications that ident++ allows in §4. We give a brief security analysis showing that ident++ in an OpenFlow network is at least as secure as firewalls in §5. Finally, we discuss some related work in §6 and conclude the paper in §7.

2. OVERVIEW

An ident++-protected network consists of ident++-enabled firewalls and end-hosts. Firewalls can be the usual standalone networking middleboxes, or can be implemented using an Ethane network [5] or an OpenFlow network [12]. On the other hand, end-hosts run an ident++ daemon as a server that receives queries on TCP port 783.

When a firewall encounters a packet belonging to a flow that is not in its rule cache, it requests additional information from *both* the source and the destination end-hosts.

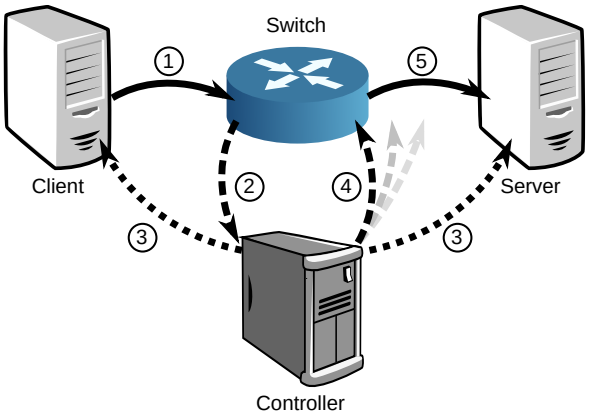

Figure 1: Overview of ident++: 1. Client initiates flow by sending packet, 2. First-hop switch forwards packet to controller 3. controller requests additional information using ident++ from both ends of flow, 4. if controller approves, it installs entries along path for flow, 5. packet proceeds to destination.

These steps are illustrated in Figure 1, where the firewall is the combination of both the controller and switch.

A flow under ident++ is defined as the 5-tuple {IP destination and source addresses, IP protocol, TCP or UDP destination and source ports}. This 5-tuple is included in the request to the ident++ daemon on the end-hosts. The ident++ daemon's response is a list of key-value pairs that includes information such as the user ID of the user that initiated the flow on the source end-host or that would receive the flow on the destination end-host, the hash and version of the executable, rules specifying flows that the receiver wants filtered, and more. ident++ does not limit the types of key-value pairs possible and allows network administrators, application developers, and users to define new keys. The additional information is used together with the flow's 5-tuple to consult the network administrator's policy and make a decision on whether to allow or drop a packet. In certain situations, such as an OpenFlow or Ethane network, the policy can use additional information such as a packet's ingress port or MAC addresses.

ident++ response and query packets can be intercepted themselves by ident++-enabled firewalls. The firewalls can answer the queries themselves or can modify response packets to insert additional information. When adding information to a response packet, a ident++-enabled firewall adds an empty line to delineate the information it has added from that supplied by upstream firewalls.

Controlled Delegation.

ident++ allows an administrator to set coarse grained access control policies, and delegate more fine-grained policies. For example, the administrator might specify that machines within a single department have unrestricted access to each other. More fine-grained policies can then be specified by department-local administrators and users. For instance, one such policy might constrain access to a code repository to developers in a specific department only.

In the rest of the paper, we will assume that firewalls are implemented using an OpenFlow network. §3 gives a more

detailed description of ident++ running in an OpenFlow network. We will give a more detailed, though not thorough, account of security in §5. Below, we briefly describe the threat model under which we operate.

Threat Model.

ident++ helps firewalls provide a first line of defense. We assume that most users are "honest". That is they do not subvert the network's security policy unless they find it that the security measures are inconvenient and think it is safe to bypass them. ident++ attempts to remove the incentives for breaking security protocols by making network security more convenient and accurate. However, users might inadvertently create security holes or allow their accounts to be compromised. Attackers might be able to compromise end-hosts, but it is more difficult to gain access as a super-user or administrator than as non-privileged users. Finally, the components of the network themselves can be attacked and compromised, though these are more difficult targets than end-hosts.

3. DESIGN

ident++ uses several ideas already encapsulated by OpenFlow [12], so it is only natural that we describe our design in the context of an OpenFlow network. We first give a brief overview of OpenFlow, then we describe our policy language *PF+=2* , then the ident++ daemon that responds to queries, and finally the ident++ controller.

3.1 OpenFlow

OpenFlow is a protocol that allows flow tables in switches and routers to be remotely managed by an OpenFlow controller. OpenFlow defines a flow as a 10-tuple {Ingress port, MAC source and destination addresses, Ethernet type, VLAN identifier, IP source and destination addresses, IP protocol, transport source and destination ports}. Note that this 10-tuple is a superset of ident++'s 5-tuple definition of a flow (see §2).

The flow table in an OpenFlow switch maps from the 10-tuple definition of a flow to an action to be taken on packets belonging to that flow. These actions include dropping the packet, forwarding it on a particular port or number of ports, or sending the packet to the OpenFlow controller. An arriving packet that does not match any of the entries in the flow table is encapsulated and sent to the OpenFlow controller for inspection. When the OpenFlow controller makes a decision regarding what to do with all packets that have the same 10-tuple flow description, it adds an entry for that flow in the switch's flow table to cache its decision. The OpenFlow controller can insert entries in switches across the network preemptively so that this process is not repeated for every switch at which the packet arrives.

In an ident++-protected OpenFlow network, the OpenFlow controller uses ident++ to query end-hosts about flows on which it needs to make a decision. The end-hosts run an ident++ daemon that responds to queries using locally stored configuration files that contain the key-value pairs to be used. The controller uses the response packet and its own configuration files to make an allow/drop decision.

3.2 Query and Response Formats

A query packet contains the flow's 5-tuple and a list of keys that the controller is interested in. The controller making the query uses the flow's destination IP address as the query's source IP address. The flow's source and destination IP addresses can then be obtained from the query's IP header while the protocol and port numbers can be found in the payload:

```
<PROTO> <SRC PORT> <DST PORT>
<key 0>
<key 1>
...
```

The list of keys in the query packet only provide a hint for what the controller needs. The response may contain additional unsolicited key-value pairs.

A reponse packet contains the flow's 5-tuple, as in the query, and a list of key-value pairs separated by line breaks. The list is broken up into sections delineated by empty lines. New sections correspond to key-value pairs from different sources. For example, a new section can exist for pairs provided by the user, the application, or the local administrator. New sections are also added by controllers augmenting the response. A response packet looks like the following:

```
<PROTO> <SRC PORT> <DST PORT>
<key 0>: <value 0>
<key 1>: <value 1>
...
<newline>
<key n>: <value n>
...
```

3.3 PF+=2

We have extended OpenBSD's PF [2] to provide a language that is intuitive, extensible, and capable of including externally provided rules. We illustrate PF+=2 first by example:

```
table <mail-server> {192.168.42.32}

block all
pass from any                              \
     with member(@src[groupID], users)     \
     with eq(@src[app-name], pine)         \
     to <mail-server>                      \
     with eq(@dst[userID], smtp)
```

The above rule blocks all flows except those going to the mail server where the sender is a member of the users group and the user is using pine. In addition, the receiver must be the smtp user.

PF provides a complete and convenient language for dealing with flows concisely, including features like lists, macros, address ranges, and port ranges. Here, we only describe a small subset, and how we extended it. In vanilla PF, rules are read in top-down order, with the last matching rule being executed. A matching rule can force its execution and bypass later rules if it contains the quick keyword. The quick keyword can help improve performance and ensure that some rules are never overridden.

Each rule starts with an action. Currently, only two are defined: pass and block, which respectively allow or deny a flow[1]. The predicates to match follow the from and to keywords and are specified as network primitives such as IP addresses or TCP/UDP ports. The predicate from any to

[1]We do not currently use the log action.

any can be replaced with the all keyword. To store a list of IP addresses, the table keyword can be used as in the example above.

To create PF+=2, we first extend PF to parse and make available the key-value pairs in ident++ responses. The responses from the source and destination are parsed to fill @src and @dst dictionaries. Since responses can be augmented by multiple controllers on the path, and keys can be repeated in different sections of the response, indexing the dictionaries will give the latest value added to the response. The latest value is the most trusted (though not necessarily the most trustworthy) because a controller can overwrite or modify any responses that it sees.

Accessing the @src or @dst dictionaries by prepending a * as in *@src[userID] returns a concatenation of the values in all sections of the response packet. The concatenated value can be used to check if a series of endorsements (such as a network path) was followed or if a value changed between networks.

PF+=2 extends PF by adding the dict and with keywords and allowing function calls that return boolean results. The dict keyword allows the definition of dictionaries. The with keyword enables predicates that match on key-value pairs returned in responses to the ident++ protocol queries.

Each with is followed by a function call that can operate on values from the @src or @dst dictionaries. Functions are user-definable and new functions can be added. The following functions are predefined:

- eq, gt, lt, gte, lte return true if first argument $=$, $>$, $<$, $\geq$, or $\leq$ second argument respectively.
- member tests if first argument is in list named by second argument.
- allowed tests if flow is allowed by rule specified in argument.
- verify tests if first argument is the correct signature for public key specified in second argument and data specified in remaining arguments.

The allowed function allows the administrator to include rules provided by others. When combined with the verify function, it effectively enables authenticated delegation. That is, it allows a user or a trusted third-party to specify "good behavior". We will give examples of how these can be used in §4.

3.4 ident++ Controller

When an OpenFlow switch cannot find a match for a packet in its flow table, it sends the packet to the ident++ controller. When the controller receives the packet, it queries the source and destination ident++ daemons for additional information. The information is then stored in the @src and the @dst dictionaries. The controller then executes the rules that are stored in its configuration files.

The controller's configuration files reside in a well known location and have the .control extension. The files are read in alphabetical order and their contents are concatenated. Some of these configuration files can be written by the administrator, while others can be provided by application developers or third-party security companies.

As was said earlier, ident++ controllers can intercept queries and responses. However, intercepted queries are not allowed to cause new queries. To respond to an intercepted query on behalf of an end-host, the controller spoofs the IP address of the end-host, sends a response itself, but does not for-

00-local-header.control

```
table <server> { 192.168.1.1 }
table <lan> { 192.168.0.0/24 }
table <int_hosts> { <lan> <server> }
allowed = "{ http ssh }"  # a macro of apps

# default deny
block all
# allow connections outbound
pass from <int_hosts> \
     to !<int_hosts> \
     keep state

# allow all traffic from approved apps
pass from <int_hosts>             \
     to <int_hosts>               \
     with member(@src[name], $allowed) \
     keep state
```

50-skype.control

```
table <skype_update> { 123.123.123.0/24 }

# skype to skype allowed
pass all                          \
     with eq(@src[name], skype)   \
     with eq(@dst[name], skype)

# skype update feature
pass from any                     \
     to <skype_update> port 80    \
     with eq(@src[name], skype)   \
     keep state
```

99-local-footer.control

```
# no really old versions of skype
block all                         \
      with eq(@src[name], skype)  \
      with lt(@src[version], 200)

# no skype to server
block from any                    \
      to <server>                 \
      with eq(@src[name], skype)
```

Figure 2: Example controller configuration files, 00. contains a number of definitions and the default "block all" policy, 50. includes rules to enable the network to allow skype traffic, 99. adds constraints to the skype rules.

```
@app /usr/bin/skype {
     name         : skype
     version      : 210
     vendor       : skype.com
     type         : voip
     requirements :                          \
         pass from any port http             \
              with eq(@src[name], skype)     \
         pass from any port https            \
              with eq(@src[name], skype)
     req-sig      : 21oir...w3eda
}
```

Figure 3: Example `ident++` daemon configuration file snippet. The file lists the key-value pairs that should be included in a response packet along with the predefined ones.

ward the query. To augment an intercepted response with additional information, the controller inserts an empty line followed by the key-value pairs it wishes to add. The controller can be configured to intercept queries and responses using additional extensions in `PF+=2` which we will not discuss in this paper.

Figure 2 shows three controller configuration files. `00-local-header.control` and `99-local-footer.control` are written by an administrator and specify site-specific rules. `50-skype.control` describes a policy that allows the application to function as intended. This last file would usually be provided by the application developer or software distributor but could also be created, extended, and shared by network administrators. While logically the files are concatenated and could be written as one file, they are split up for manageability.

3.5 `ident++` Daemon

End-hosts run a simple userspace `ident++` daemon that responds with the key-value pairs to controller queries. The daemon can answer queries both when the end-host is the source and when it is a destination that has yet to accept a connection.

The `ident++` daemon gets key-value pairs in two ways: From configuration files statically and from the application using a flow dynamically. Like the controller, the `ident++` daemon has a number of configuration files residing in well known locations on the end-host. These configuration files contain the key-value pairs to be returned in a response. Some configuration files can be modified by users to insert their inputs to the system, while others reside in the system's configuration directory (such as "/etc/identxx" for Linux) and are only modifiable by the local end-host administrator. These configuration files can be written by the user or administrator or can be obtained from the application developers, software distributor (in a GNU/Linux distribution, for example), or by another trusted third-party. The configuration files can contain a signature key-value pair authenticating other key-value pairs. Figure 3 shows an example.

The last source for key-value pairs is the application using the flow. The application can provide key-value pairs to the `ident++` daemon at run-time. This mechanism can be used by a web browser, for example, to distinguish between flows that were initiated in response to user mouse clicks

research-app.conf

```
@app /usr/bin/research-app {
  name         : research-app
  # research-apps only talk to each other
  requirements :                              \
     block all                                \
     pass all                                 \
          with eq(@src[name], research-app)   \
          with eq(@dst[name], research-app)   \
  req-sig      : jsdfr...ipox7
}
```

Figure 4: Snippet from the `ident++` daemon configuration file used for the research application.

30-research.control

```
dict <pubkeys> {               \
     research : sk3ajf...fa932 \
     admin    : a923jx...a12kz \
}

# Allow only researchers to run applications
# and only access their own machines.
# Let reserachers specify what their apps need.
pass from <research-machines>             \
     with member(@src[groupID], research) \
     to !<production-machines>            \
     with member(@dst[groupID], research) \
     with allowed(@dst[requirements])     \
     with verify(@dst[req-sig],           \
                 @pubkeys[research],      \
                 @dst[exe-hash],          \
                 @dst[app-name],          \
                 @dst[requirements])
```

Figure 5: Example rule that allows researchers to run any application on their machines as long as it conforms to their own rules, and does not access production machines.

and others that are not requested by a user. These pairs are sent to the `ident++` daemon via a Unix domain socket.

The `ident++` daemon uses the 5-tuple in the query packet to find the process ID and user ID associated with the flow using techniques similar to `lsof`. The daemon uses the process ID to find the file name of the process's executable image. The file name is then used to locate the configuration files to which it corresponds. The daemon reads the configuration files and the relevant key-value pairs are placed into the corresponding sections in the response.

4. APPLICATIONS

Here we describe a few scenarios where `ident++` allows delegation, simplifies network management, and allows more flexible policies.

Delegation to Users.

In some cases, especially in research networks, users run their own applications and servers. It is tedious and time-

thunderbird.conf

```
@app /usr/bin/thunderbird {
  name         : thunderbird
  type         : email-client
  rule-maker   : Secur
  requirements :                              \
     block all                                \
     pass from any                            \
          with eq(@src[name], thunderbird)    \
          to   any                            \
          with eq(@dst[type], email-server)
  req-sig      : kaj7v...as1d3
}
```

Figure 6: Snippet from the ident++ daemon configuration file for thunderbird supplied by a third-party.

consuming to ask the network administrator to open up new ports in the firewall every time a new application needs to be tested or a user installs a new application.

The user specifies the application's network access requirements in the application's `ident++` daemon configuration file as in Figure 4. In this example, the research applications are only allowed to communicate with other research applications on non-production machines. The user signs the requirements along with the application name and executable and puts the signature in the configuration file so that an attacker cannot modify them or create new application configuration files.

Figure 5 shows how a user can be given clearance to escalate her network privileges when needed without asking the network administrator. The controller checks that the flow to which the packet belongs is allowed by the signed application requirements that are in the `ident++` receiver response packets using the `allowed` keyword, and by the administrator's policies using the `from` and `to` keywords.

Trust Delegation.

In the previous example, we showed how an administrator can delegate control of part of the network to users and allow them to run whatever they like, while still conforming to the administrator's coarse-grained policy. In this example, we show how delegation could be used to allow users to run any applications for which a third trusted party provides rules. This third party can be a security company whose business is to publish firewall rules for applications.

The `ident++` daemon's configuration file for the thunderbird application is shown in Figure 6. This file was provided be a third party security company called `Secur`. The `ident++` daemon on that machine responds to queries on a flow started by thunderbird by including the key-value pairs supplied by Secur, including Secur's signature on the rule for thunderbird. Figure 7 shows the rule responsible for checking that an application has been approved by Secur.

User and Application-specific Rules.

The rule in Figure 8 allows a network administrator to restrict access to important services and applications. The rule only permits `System` users access to the `Server` service and blocks any access from the Internet at large. In addition, the destination operating system must have the latest patch

30-secur.control

```
dict <pubkeys> {                \
     Secur    : a923jx...a12kz \
}

# Allow users to run any applications approved
# by Secur and following rules Secur provides
pass from any                                \
     with eq(@src[rule-maker], Secur)        \
     with allowed(@src[requirements])        \
     with verify(@src[req-sig],              \
                 @pubkeys[Secur],            \
                 @src[exe-hash],             \
                 @src[app-name],             \
                 @src[requirements])         \
     to any
```

Figure 7: Example rule allowing users to run any application as long as it conforms to rules specified by Secur, a third-party security company.

installed. This rule can be used to stop the Conficker [3] worm that attacks the Server service in Windows.

Network Collaboration.

Consider the case of two remote branches of the same enterprise with end-hosts communicating over the Internet. `ident++` can be used to filter traffic at one branch that the other branch would not accept. This can be used to minimize traffic between the branches if the link is a bottleneck. The controller modifies responses to queries and adds signed rules that specify what the network at a branch is willing to accept or it can request a flow be explicitly dropped.

Incremental Benefit.

It is not necessary that all components of the network support `ident++` for its benefits to be reaped. If only end-hosts implement it, it can be used by servers to distinguish users behind a NAT, or users on a single machine. On the other hand, if the controllers implement it but not the end-hosts, controllers can answer some queries on behalf of end-hosts, and networks can still collaborate.

5. SECURITY IMPLICATIONS

`ident++` on OpenFlow is not designed to be more secure than non-`ident++` firewalls. In this section we study the security of an `ident++`-enabled network, and describe the maximum damage an attacker can inflict by compromising each component of the network. We compare the potential damage done in a network protected by `ident++`-enabled firewalls with that in a a network protected by vanilla firewalls.

5.1 Controllers

The most powerful component in the network is the `ident++` controller. If the controller is compromised, an attacker can disable all protection in the network. While non-`ident++` firewalls are not usually managed by a centralized controller, they usually use the same password or are clones of each other allowing them to be compromised in the same way. Protecting the controller is as important as protecting the

10-user-rules.control

```
# default block everything
block all

# only allow ‘‘system’’ users in the LAN
pass from <lan>                                  \
     with eq(@src[userID], system)               \
     to <lan>                                    \
     with eq(@dst[userID], system)               \
     with eq(@dst[name], Server)                 \
     with includes(@dst[os-patch], MS08-067)
```

Figure 8: Example rule to allow the “System” user access to the “Server” service inside the LAN and only if the destination operating system has the latest patch.

administrative interface of regular firewalls in a network, and is comparable in difficulty.

5.2 Switches

Similar to compromising a single non-`ident++` firewall, compromising a single `ident++`-enabled switch can disable the protection it affords. Any traffic would be able to pass through the switch without regulation. However, compromising a switch does not necessarily enable the compromise of the controller.

5.3 End-Hosts

As in a vanilla firewall-protected network, a compromised end-host in a `ident++`-enabled network, allows an attacker with administrative priviliges to communicate with any other end-host it was able to communicate with before the compromise. The attacker would gain control of the `ident++` daemon and can send false `ident++` responses.

It would seem that the attacker, in addition, can request that the controller allow new flows to other destinations when control is delegated. However, a request would require the approval of the user in whose name the request is made because the request needs to be signed with the user's private key.

5.4 Users

A compromised application can masquerade as any other application. The attacker can then gain the network privileges associated with the user and any application the user can run.

One way compromised processes can achieve this on a Unix-like system is to `exec` other more privileged processes then use the `ptrace` system call to subvert the `exec`'d process. For stronger isolation the administrator can mark user applications as `setgid` with a group that does not have any file access. Processes executing with the `setgid` bit set in this manner are protected against `ptrace`. The administrator can arbitrarily subdivide and isolate user processes this way.

If a user's application is compromised then the user's privileges can be abused under `ident++`. This situation, however, is an improvement over today's firewalls. Current firewalls cannot identify users in traditional systems, so a compromise of a user's application in today's network grants the attacker all the network privileges associated with all users. On the other hand, in an `ident++`-protected network, as in an Ethane protected network, compromising one user account does not allow the attacker to abuse another user's priviliges.

Currently, hosts emulate users as protection domains by requiring superuser privileges to bind to ports below 1024. This emulation is usually implemented by forking these processes as the superuser before changing to a user with lower privilege. The superuser is effectively endorsing applications bound to privileged ports. `ident++` makes these endorsements explicit, sending them to the controller, and allowing the controller to make a decision on the administrator's behalf.

6. RELATED WORK

Distributed firewalls [9] centralize the policy, and distribute enforcement to firewalls implemented on the end-host. Distributed enforcement to the end-hosts frees the network from relying on topology for protection and allows a host to use additional information, as in `ident++`, to make the policy decision. Unfortunately, the distributed firewalls approach suffers from a number of problems. First, if enforcement is done only at the receiving end-host in this way, the end-host can become vulnerable to denial of service attacks. Second, a compromised end-host effectively has no protection. The central administrator's policies are completely bypassed. Finally, distributed firewalls do not solve the delegation problem and their policies are written in terms of incidental flow properties and network primitives rather than principals. By contrast, `ident++` centralizes the policy as in distributed firewalls, but only uses end-hosts to extract additional information. It keeps enforcement in the network where it can be done at line-rate and closer to the source.

Ethane [5] provides administrators with centralized control of network flows in an enterprise network. However, it forces the administrator to make security decisions based on the source and destination's physical switch ports and network primitives, and not on any application-level information.

The ident protocol [14] was originally used by one end-point of a TCP connection to identify the user responsible for the remote end-point. This information was typically used for auditing purposes, such as tracking down a compromised account that was sending spam. This paper expands on the idea of the ident protocol to provide much richer information about connections; to allow intermediate network elements to provide additional information about connections; and to use this information for firewall decisions in the network.

VPNs and VLANs are used extensively in enterprise networks to separate “trusted” traffic from the rest of the Internet, or to separate different applications or workloads on the same enterprise network. Using VLANs and VPNs require users and administrators to partition the traffic on each client machine ahead of time, or to assign switch ports, and thus entire machines, to specific VLANs. `ident++` lets users and administrators control network use at a finer granularity, potentially allowing controlled interaction between different applications, which would be difficult to achieve if each application were using its own VLAN. At the same time, we expect `ident++` would be used in conjunction with VPNs and VLANs, which provide stronger enforcement in

situations where the client machines might be malicious or compromised.

DStar [15] allows individual machines to communicate information flow restrictions for the contents of network messages to each other. While DStar allows two cooperating machines to isolate the network communication of multiple applications running on each machine from one another, it does not export higher-level information about the flows to the network administrator, and thus does not allow the network administrator to specify firewall rules for applications.

Multi-level secure networks [11, 13, 7, 4] allow sharing a single physical network between different applications that should not be able to leak data to one another. These systems provide stronger isolation than `ident++`. Much like VLANs, these systems require each machine to categorize its network messages ahead of time, e.g. "secret", "top secret", etc, leaving the network administrator with no information as to what application originated any particular connection. While this model may be appropriate for military applications, `ident++` gives the network administrator much more information about the traffic on the network that can be used as a basis for filtering.

7. CONCLUSION

Delegating control and trust to end-hosts and users allows flexible and more powerful management of an enterprise network. With network security heading towards more centralized control, administrators will want to delegate some control back to users and end-hosts. `ident++` is a simple proposal for a system that gives administrators the ability to delegate control and to override, audit, and revoke the delegation when necessary. `ident++` enables policies and solutions that are otherwise not possible to achieve.

`ident++` allows a policy to be written in terms of principals that security is concerned with—users, applications, versions, etc. It allows a trusted third-party to specify rules which a user can provide without worrying whether the application is behaving correctly or not.

While some might consider security paramount to user convenience, making security too tight and inconvenient can hamper productivity and cost an enterprise valuable time and employee morale. On the other hand, delegating control under close supervision allows the administrator to treat security problems as the less common case, and allows her to retain full power over the network.

Acknowledgments

This research was performed under an appointment to the U.S. Department of Homeland Security (DHS) Scholarship and Fellowship Program, administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and DHS. ORISE is managed by Oak Ridge Associated Universities (ORAU) under DOE contract number DE-AC05-06OR23100. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of DHS, DOE, or ORAU/ORISE.

8. REFERENCES

[1] Aruba networks. http://www.arubanetworks.com. Last accessed on 3/18/2009.

[2] PF: The OpenBSD Packet Filter, Jan. 2008. http://www.openbsd.org/faq/pf/.

[3] Virus alert about the win32/conficker.b worm. http://support.microsoft.com/kb/962007, Mar. 2009.

[4] J. P. Anderson. A unification of computer and network security concepts. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 77–87, Oakland, CA, 1985.

[5] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. *ACM SIGCOMM Computer Communications Review (CCR)*, 37(4):1–12, 2007.

[6] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. SANE: a protection architecture for enterprise networks. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, Berkeley, CA, USA, 2006. USENIX Association.

[7] D. Estrin. Non-discretionary controls for inter-organization networks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 56–61, Oakland, CA, 1985.

[8] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4d approach to network control and management. *ACM SIGCOMM Computer Communications Review (CCR)*, 35(5):41–54, 2005.

[9] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith. Implementing a distributed firewall. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 190–199, New York, NY, USA, 2000. ACM.

[10] D. A. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. *ACM SIGCOMM Computer Communications Review (CCR)*, 38(4):51–62, 2008.

[11] J. McHugh and A. P. Moore. A security policy and formal top-level specification for a multi-level secure local area network. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 34–39, Oakland, CA, 1986.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communications Review (CCR)*, 38(2):69–74, 2008.

[13] D. P. Sidhu and M. Gasser. A multilevel secure local area network. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 137–143, Oakland, CA, 1982.

[14] M. C. St. Johns. Identification protocol. RFC 1413, Network Working Group, February 1993.

[15] N. Zeldovich, S. Boyd-Wickizer, and D. Mazières. Securing distributed systems with information flow control. In *Proceedings of the 5th Symposium on Networked Systems Design and Implementation*, pages 293–308, San Francisco, CA, April 2008.