Implementing SCADA Security Policies via Security-Enhanced Linux

Ryan Bradetich
Schweitzer Engineering Laboratories, Inc.

Paul Oman
University of Idaho, Department of Computer Science

Presented at the
10th Annual Western Power Delivery Automation Conference
Spokane, Washington
April 8–10, 2008

Implementing SCADA Security Policies via Security-Enhanced Linux

Ryan Bradetich, *Schweitzer Engineering Laboratories, Inc.*
Paul Oman, *University of Idaho, Department of Computer Science*

***Abstract*—Substation networks have traditionally been isolated from corporate information technology (IT) networks. Hence, the security of substation networks has depended heavily upon limited access points and the use of point-to-point supervisory control and data acquisition (SCADA)-specific protocols. With the introduction of Ethernet into substations, pressure to reduce expenses and provide Internet services to customers has many utilities connecting their substation networks and corporate IT networks, despite the additional security risks. The Security-Enhanced Linux SCADA proxy was introduced as an alternative method for connecting SCADA and corporate IT networks by serving as a check valve between the SCADA system and the IT network. The primary purpose of the Security-Enhanced Linux SCADA proxy was to prohibit direct access and interference from the corporate IT network to the SCADA network, while still providing read-only access to specific SCADA data.**

This paper extends the prior research on the Security-Enhanced Linux SCADA proxy with:

- **An in-depth look at how the security policy enforces the one-way communication for proxying specific SCADA data from a protective relay to engineers on the corporate IT network.**
- **The integration of settings into the security policy on the Security-Enhanced Linux SCADA proxy (network settings, syslog servers, etc.)**
- **The use of physical contact inputs to switch dynamically between predefined behaviors in the security policy.**

I. Introduction

Supervisory control and data acquisition (SCADA) systems and corporate information technology (IT) networks evolved independently and have, until recently, remained isolated from each other. To reduce costs and capitalize on common standards, vendors and business managers are connecting SCADA systems with corporate IT networks. Current SCADA security literature is advocating traditional IT security solutions, such as strong passwords, encrypted communications, and firewalls [1]. No assurance exists that these mechanisms can provide adequate security for critical real-time control networks.

SCADA systems operate in a fundamentally different way than corporate IT networks. SCADA systems manage critical infrastructures such as the transmission and distribution of electricity, while corporate IT networks manage business. SCADA system outages may result in environmental damage and/or the loss of human life [2]. Outages on the corporate IT network are generally financial and localized to a specific corporation. Unsurprisingly, the protocols used on SCADA and corporate IT networks are also fundamentally different. SCADA protocols provide efficient, deterministic communications between devices [3] [4]. Corporate IT protocols generally provide reliable communications over a shared communications channel.

Security-Enhanced Linux was initially developed by the National Security Agency to introduce a mandatory access control (MAC) solution into the Linux® kernel [5]. The original proof-of-concept Security-Enhanced Linux SCADA proxy introduced in [6] and [7] provided a baseline system. The purpose of that baseline was to explore the feasibility of using the MAC security model to maintain the logical network isolation between the SCADA and the corporate IT networks, while still providing SCADA data to authorized corporate IT users.

To understand the improvements to the Security-Enhanced Linux SCADA proxy presented in this paper, see Fig. 1, which illustrates the high-level application domains and communications paths for the original Security-Enhanced Linux SCADA proxy.

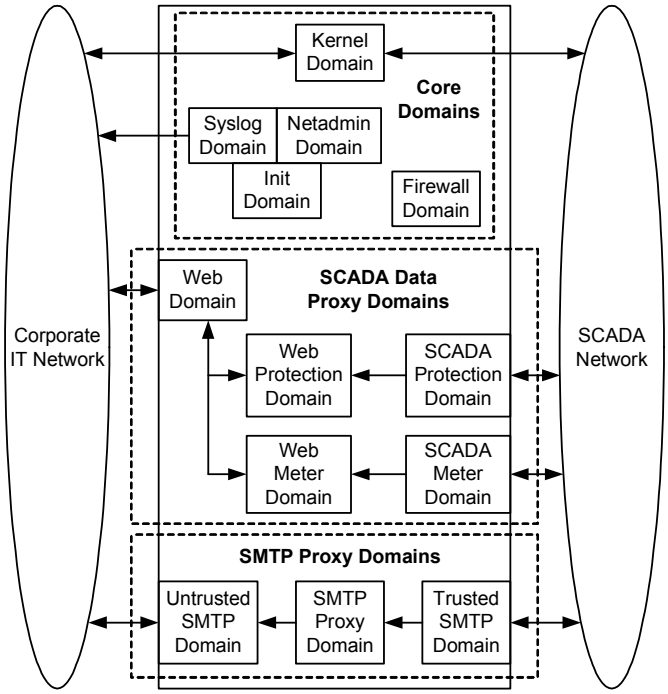

Fig. 1. Original Security-Enhanced Linux SCADA Proxy—Application Domains and Communications Paths

The original Security-Enhanced Linux SCADA proxy functionality was divided into three application groups: core

domains, SCADA data proxy domains, and SMTP proxy domains. The core domains are responsible for basic Linux platform operations. The SCADA data proxy domains are responsible for providing SCADA data to authorized users on the corporate IT network. The SMTP proxy domains are responsible for the one-way transfer of SMTP (email) from the SCADA network to the corporate IT network.

The following three major architectural changes were made to the original Security-Enhanced Linux SCADA proxy to prepare it for the research presented in this paper.

1. The Security-Enhanced Linux SCADA proxy was ported to a new platform, which supports physical contact inputs and physical contact outputs.
2. An additional web server was set up on the SCADA network interface. This web server is referred to as the trusted web server and is used for updating settings on the Security-Enhanced Linux SCADA proxy.
3. The SMTP proxy domains were not relevant to the research presented in this paper. To simplify the policy and application domain interactions, this application group was removed.

Fig. 2 provides the new high-level application domains and communications paths used in the updated Security-Enhanced Linux SCADA proxy for this paper.

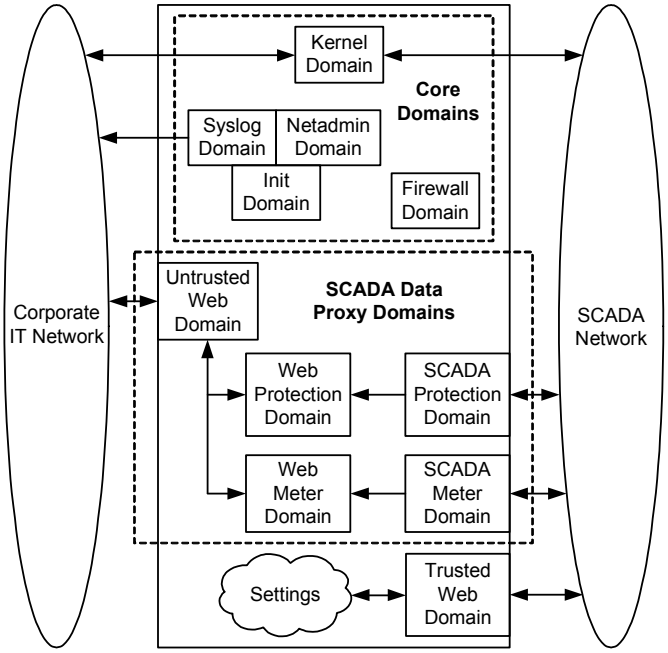

Fig. 2. Updated Security-Enhanced Linux SCADA Proxy—Application Domains and Communications Paths

This paper has two purposes: to explore how user settings can be securely integrated into the Security-Enhanced Linux SCADA proxy and to provide a detailed look at how the Security-Enhanced Linux security policy provides the one-way communications path via the SCADA data proxy domains. For this paper, Section II describes the basics of the Security-Enhanced Linux policy language. Section III exposes the Security-Enhanced Linux policy language used by the SCADA data proxy domain. Section IV and Section V describe how user settings were integrated into the Security-Enhanced Linux SCADA proxy.

II. Security-Enhanced Linux Policy Basics

The Security-Enhanced Linux security policy provides the MAC security model for the Security-Enhanced Linux SCADA proxy. Although the standard Linux discretionary access control (DAC) model is not discussed much in this paper, the MAC security model does not replace the standard Linux DAC model. The MAC security model resource checks are performed after the standard Linux DAC resource checks. This is important for the following reasons:

- Permissions to resources (i.e., files, directories, etc.) must pass both the Linux DAC and MAC checks.
- Security-Enhanced Linux generates audit messages when a policy violation occurs. The Security-Enhanced Linux SCADA proxy takes advantage of this situation by carefully matching the privileges in the DAC and MAC security models. This allows the Security-Enhanced Linux SCADA proxy to generate policy violation audit messages (which are sent to a remote host via syslog) to indicate that the Security-Enhanced Linux security proxy may be under attack by an intruder.

The following two sections provide background information for the Security-Enhanced Linux security policy statements used in this paper. The policy language statement use for access control checks is described in Section A. Section B describes how Security-Enhanced Linux controls the creation and relabeling of processes in different application domains.

A. Access Vector (AV) Rules

The default Security-Enhanced Linux security policy is to deny access to all resources. The Security-Enhanced Linux policy language uses AV rules to specify permissions to resources.

```
rule source(s) target(s):class(es) permission(s);
```

Fig. 3. Security-Enhanced Linux AV Rule Syntax

The five elements of the AV rule syntax are discussed below:[1]

1. *rule* – Security-Enhanced Linux currently supports four AV rule types. This paper only describes the *allow* and *neverallow* rule types.[2] The *allow* rule type permits the specified access between the sources and targets. The *neverallow* rule type is a policy compile-time check to ensure the specified access between the sources and targets is never permitted.
2. *source(s)* – Security-Enhanced Linux adds labels to all processes running on the Linux system. These labels are referenced as the source(s) in the AV rules. Fig. 4

[1] The Security-Enhanced Linux security policy supports multiple elements within a single rule but requires multiple elements to be enclosed within { }.

[2] The other two rule types deal with generating additional and suppressing audit messages.

illustrates the additional Security-Enhanced Linux label attribute in addition to the user identifier (UID) and group identifier (GID) attributes used in the standard Linux DAC security model.

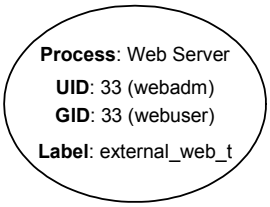

Fig. 4. Labeled Process

3. *target(s)* – Security-Enhanced Linux also adds labels to all resources on the Linux system. These labels are referenced as the target(s) in the AV rules. Fig. 5 illustrates the additional Security-Enhanced Linux label attribute in addition to the UID, GID, and permission label attributes used in the standard Linux DAC security model.

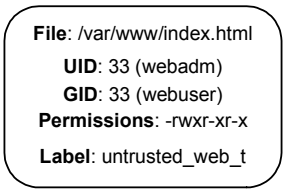

Fig. 5. Labeled File

Security-Enhanced Linux also labels resources that are not traditionally used in the DAC security model. Fig. 6 illustrates the additional Security-Enhanced Linux label attribute added to a network interface.

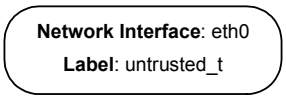

Fig. 6. Labeled Network Interface

4. *class(es)* – Security-Enhanced Linux represents kernel resources as classes.[3] Example kernel resources protected by Security-Enhanced Linux include files, directories, network interfaces, network communications, and System V IPC communications.[4]
5. *permission(s)* – The permission element specifies the access permissions for the specified element class(es). Each element class defines the set of available permissions in the Security-Enhanced Linux security policy. For example, the 2.6.23.9 Linux kernel defines the following permissions for the file element class: *append*, *create*, *execute*, *getattr*, *ioctl*, *link*, *lock*, *mounton*, *quotaon*, *read*, *relabelfrom*, *relabelto*, *rename*, *setattr*, *swapon*, *unlink*, and *write*.

Fig. 7 provides an example Security-Enhanced Linux policy statement that permits processes with the *untrusted_web_t* label *read* access to files with the *lib_t* label.

```
allow untrusted_web_t lib_t:file read;
```

Fig. 7. Example Policy #1

Fig. 8 provides an example Security-Enhanced Linux policy statement with multiple target and permission elements. This policy statement gives the *untrusted_web_t* domain *read*, *getattr*, and *search* permissions to directories labeled with the *root_t* and *system_t* labels.[5]

```
allow untrusted_web_t { root_t system_t }
      :dir { read getattr search };
```

Fig. 8. Example Policy #2

The *neverallow* policy statements do not remove permissions from the security policy; instead, they generate policy compiler errors [5]. The *neverallow* rule supports two additional syntax operators, the wildcard (*) and complement (~). Fig. 9 illustrates a Security-Enhanced Linux policy statement that prevents any process with the *untrusted_web_t* label from accessing any resources on any network interface without the *untrusted_t* label.

```
neverallow untrusted_web_t ~untrusted_t
                :netif *;
```

Fig. 9. Example Policy #3

B. Domain Transitions

The AV rules specify the permissions each domain has to the Linux system resources. Security-Enhanced Linux also supports transitioning a process from one application domain into a different application domain. Dynamic domain transitions and default domain transitions are the two types of domain transitions supported.

The dynamic domain transition privilege was added to Security-Enhanced Linux primarily for compatibility with other systems. Dynamic domain transition permits the process to execute arbitrary code in a different domain, destroying the separation between application domains. The dynamic domain transition is not recommended for use in Security-Enhanced Linux policies and is not discussed further in this paper [5].

Default domain transitions occur when an existing domain creates a new Linux process and the Security-Enhanced Linux security policy is configured to automatically relabel the process to the new domain. To configure the Security-Enhanced Linux security policy for default domain transitions, the following four requirements must be met:

1. The current domain must be permitted to execute the domain entry point file.[6]
2. The domain entry point file must have the *entrypoint* file permission.
3. The current domain process must have permission to *transition* to the new domain.
4. A *type_transition* policy statement must exist to change the process label when executing the domain entry point file.

[3] The 2.6.23.9 Linux kernel provides 42 classes. Security-Enhanced Linux also provides support for nonkernel classes. This paper does not address any of the nonkernel classes.

[4] Examples of System V IPC communications include semaphores, message queues, and shared memory.

[5] The *getattr* permission is used to protect access to specific attributes of an object, such as access modes [5].

[6] The file executed to launch the new Linux process is referred to as the domain entry point file.

Fig. 10 provides the Security-Enhanced Linux policy statements required for the *init_t* domain to transition into the *untrusted_web_t* domain.

```
# Domain transition rule.
type_transition init_t untrusted_web_exec_t
        :process untrusted_web_t;

# Mark the domain entry point file.
allow untrusted_web_t untrusted_web_exec_t
        :file entrypoint;

# Execute permissions on domain entry point.
allow init_t untrusted_web_exec_t
        :file execute;

# Permit the process transition.
allow init_t untrusted_web_t
        :process transition;
```

Fig. 10. Required Policy Statements for Default Domain Transitions

Fig. 11 shows the default domain transitions configured into the security policy for the Security-Enhanced Linux SCADA proxy.

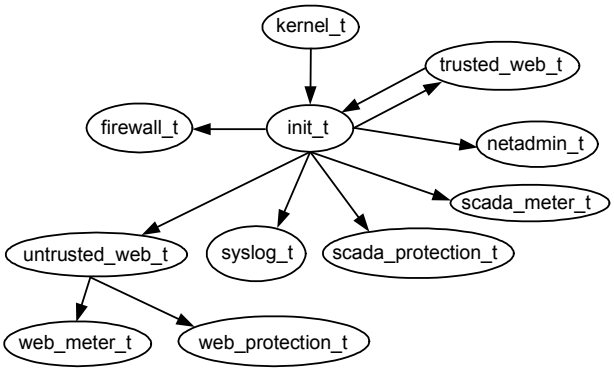

Fig. 11. Security-Enhanced Linux SCADA Proxy Domain Transitions

Analysis of the default domain transitions in the Security-Enhanced Linux security policy exposes which domains an intruder could potentially compromise during an attack. For the Security-Enhanced Linux SCADA proxy, the intruder is assumed to scan all ports and services but focus the attack on the *untrusted_web_t* domain.

After reviewing the domain transition diagram for the original Security-Enhanced Linux SCADA proxy, it is reasonable to assume the intruder could compromise the *untrusted_web_t*, *web_meter_t*, and *web_protection_t* domains.[7] To prevent any additional domain transitions from the *untrusted_web_t* domain, this implementation of the Security-Enhanced Linux SCADA proxy provides a web server on the SCADA network interface. This web server runs in the *trusted_web_t* domain and is used for changes in the electronic device settings on the Security-Enhanced Linux SCADA proxy.

III. SCADA Data Proxy Domains

The primary purpose of the Security-Enhanced Linux SCADA proxy is to logically isolate the SCADA and corporate IT networks, while still providing SCADA data to authorized users on the corporate IT network. The SCADA application domains (*scada_meter_t* and *scada_protection_t*) are responsible for collecting data from SCADA devices and storing these data locally on the Security-Enhanced Linux SCADA proxy. The web application domains (*untrusted_web_t*, *web_meter_t*, and *web_protection_t*) are responsible for presenting these SCADA data in useful formats for authorized users on the corporate IT network. The original Security-Enhanced Linux SCADA proxy presented a hypothetical scenario, modeled in the test environment, to illustrate how the Security-Enhanced Linux SCADA proxy is able to meet this objective. This paper uses the same hypothetical scenario and test environment to expose the Security-Enhanced Linux security policy for the SCADA data proxy domains enforcing a strict one-way communications path. A short description of the hypothetical scenario and test environment is provided for background information.

As described in [7], the test environment is modeled after a hypothetical scenario of an electric utility factory interconnect. The factory has several wind turbines to help offset the cost of power from the electric utility. The power generated from the wind turbines is insufficient to power the factory during daily production runs, so power from the electric utility is still required. At night, when the power consumption is much lower, the factory sells the excess power generated from the wind turbines back to the electric utility. Fig. 12 illustrates the one-line circuit diagram for the hypothetical test environment.

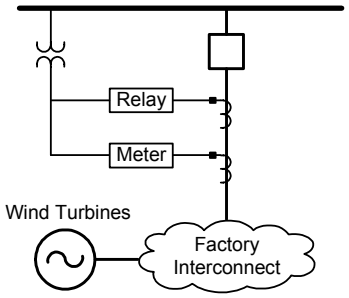

Fig. 12. Test Environment—One-Line Diagram

Fig. 13 illustrates the communications paths used in the test environment. The adaptive multichannel source (AMS) is configured to provide simulated voltages and currents to both the protective relay and the meter. Periodically, the AMS will also simulate power system faults in the test environment.

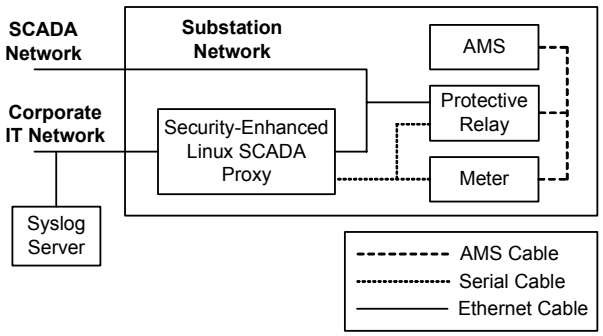

Fig. 13. Security-Enhanced Linux SCADA Proxy Communications Diagram

[7] The domain transition diagram for the original Security-Enhanced Linux SCADA proxy can be found in Figure 6.6 in [7], in which *untrusted_web_t* is labeled as *web_t*.

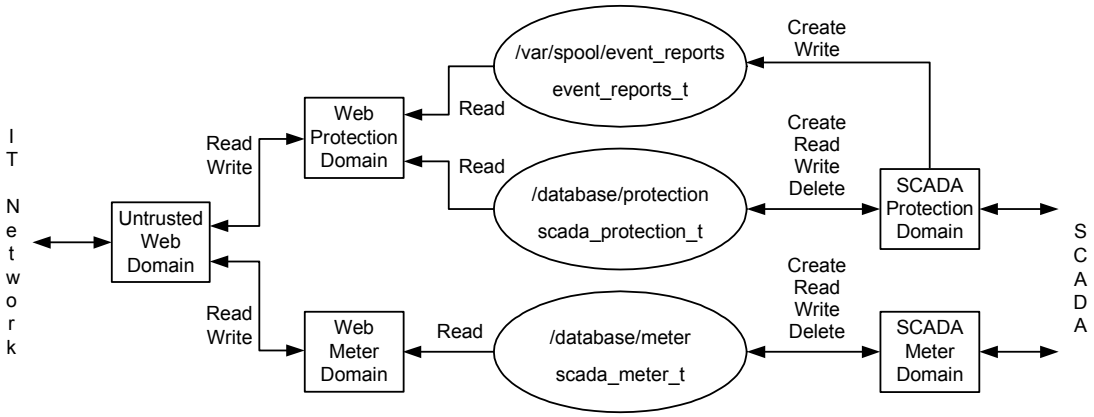

Fig. 14. SCADA Data Proxy—One-Way Communications Path

Fig. 14 maps the one-way communications path and the separation of SCADA data into a high-level security policy. The Security-Enhanced Linux application domains are represented as boxes and are discussed in additional detail in the sections below. The ovals represent SCADA data storage on the file system and are responsible for enforcing the one-way communication.

A. SCADA Meter Domain

The SCADA meter domain is responsible for collecting both instantaneous data and load profile records from the revenue meter. The meter data collection process is the only process labeled with the *scada_meter_t* label. This process connects to the revenue meter via a serial connection using a device file.[8] Fig. 15 provides a detailed look at how the SCADA meter domain participates in the one-way communications path through the SCADA data proxy domains.

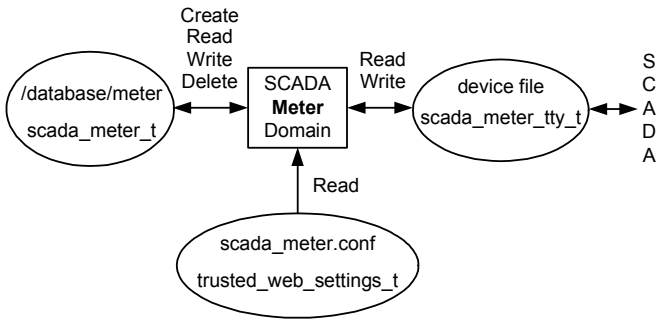

Fig. 15. SCADA Meter Domain Permissions

The *trusted_web_t* domain has privileges to relabel the user-specified device file with the *scada_meter_tty_t* label. The *trusted_web_t* domain also has privileges to update the *scada_meter.conf* configuration file. On startup, the meter collection application reads the *scada_meter.conf* configuration file to determine which device file to use for connecting to the revenue meter. Fig. 16 provides the Security-Enhanced Linux policy statements to ensure only the *trusted_web_t* and *scada_meter_t* domains are able to access device files labeled with the *scada_meter_tty_t* label.

```
# meter collection has read/write access to the device file.
allow scada_meter_t scada_meter_tty_t
        :chr_file { read write ioctl };

# trusted web has relabel access to the device file.
allow trusted_web_t scada_meter_tty_t
        :chr_file { getattr relabelfrom relabelto };

# prohibit all other access to the device file.
neverallow ~{ scada_meter_t trusted_web_t } scada_meter_tty_t
        :chr_file *;
neverallow scada_meter_t scada_meter_tty_t
        :chr_file ~{ read write ioctl };
neverallow trusted_web_t scada_meter_tty_t
        :chr_file ~{ getattr relabelfrom relabelto };
```

Fig. 16. SCADA Meter Domain—Device File Policy Statements

Fig. 17 provides the Security-Enhanced Linux policy statement to ensure only the *trusted_web_t* domain has permission to write to the *scada_meter.conf* configuration file.

```
neverallow ~trusted_web_t trusted_web_settings_t
        :file ~{ read getattr };
```

Fig. 17. Settings Configuration File Policy Statements

The data collected from the revenue meter is stored in an SQLite database. The SQLite database files are stored in the */database/meter* directory and labeled with the *scada_meter_t* label. To maintain database integrity during updates, the SQLite database engine creates a temporary file in the */database/meter* directory. To prevent temporary file security attacks, only processes in the *scada_meter_t* domain are permitted to create and remove files from the */database/meter* directory. Fig. 18 provides the Security-Enhanced Linux policy statements to ensure only the *scada_meter_t* domain has permission to create, remove, read, and write to files in the */database/meter* directory. Fig. 18 also shows that only the *web_meter_t* and *init_t* domains have permission to read the SCADA data stored in this SQLite database.

[8] A device file is a special file that allows user-space applications to interact with kernel-controlled resources (i.e., serial ports).

```
# meter collection has create/read/write/delete permissions
allow scada_meter_t scada_meter_t
        :file { create read getattr write unlink lock };

# web meter has read permissions
allow web_meter_t scada_meter_t
        :file { read getattr lock };

# init domain requires read permissions
allow init_t scada_meter_t:file read;

# prohibit all other access to the SCADA data
neverallow ~{ scada_meter_t web_meter_t init_t } scada_meter_t
        :file *;
neverallow scada_meter_t scada_meter_t
        :file ~{ create read getattr write unlink lock };
neverallow web_meter_t scada_meter_t
        :file ~{ read getattr lock };
neverallow init_t scada_meter_t
        :file ~read;
```

Fig. 18. SCADA Meter Domain—Database File Permissions

The Security-Enhanced Linux policy statements provided in this section show how the SCADA meter domain participates in the one-way communications design of the Security-Enhanced Linux SCADA proxy. Fig. 16 shows that the *scada_meter_t* domain is the only application domain permitted to communicate with device files with the *scada_meter_tty_t* label. Fig. 18 shows that the *scada_meter_t* domain is the only application domain with update privileges to the SCADA data stored in the SQLite database in the */database/meter* directory. Fig. 19 provides the final policy statement to ensure all communications paths are prohibited between the application domains only connected by white space in Fig. 14.

```
neverallow { untrusted_web_t web_protection_t
                scada_protection_t }
        { scada_meter_t scada_meter_exec_t
        scada_meter_tty_t }:
        { blk_file chr_file dir fd fifo_file file filesystem
        lnk_file sock_file association key_socket netif
        netlink_audit_socket netlink_dnrt_socket
        netlink_ip6fw_socket netlink_kobject_uevent_socket
        netlink_nflog_socket netlink_route_socket sem shm
        netlink_selinux_socket netlink_socket msg msgq
        netlink_tcpdiag_socket netlink_xfrm_socket
        node packet_socket rawip_socket socket tcp_socket
        udp_socket unix_dgram_socket unix_stream_socket ipc
        capability passwd pax process security system } *;
```

Fig. 19. SCADA Meter Domain—White Space Verification

B. SCADA Protection Domain

The SCADA protection domain is responsible for collecting both instantaneous data and event reports from the protective relay. This domain has two processes running in it, the relay data collection application and the event reports collection application. Both applications are labeled with the *scada_protection_t* label. The relay data collection application connects to the protective relay via a serial connection using a device file. The event report collection application connects to the protective relay via a network connection. Fig. 20 provides a detailed look at how the SCADA protection domain participates in the one-way communications path through the SCADA data proxy domains.

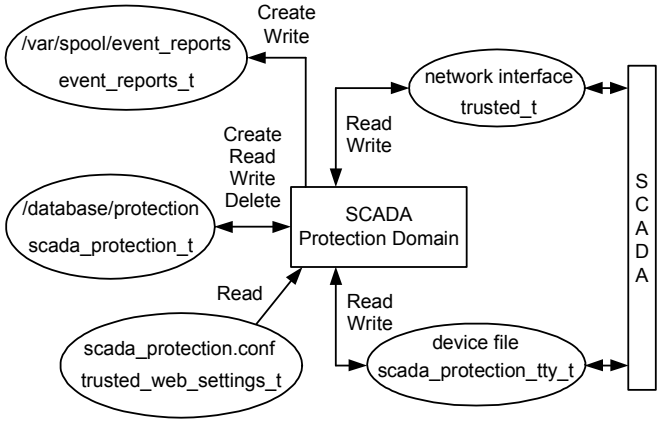

Fig. 20. SCADA Protection Domain Permissions

The *trusted_web_t* domain has privileges to relabel the user-specified device file with the *scada_protection_t* label but does not have privileges to relabel network interfaces. The *trusted_web_t* domain also has privileges to update the *scada_protection.conf* configuration file. On startup, both the relay data collection and event report collection applications read the *scada_protection.conf* configuration file to determine the device file and IP address to use for connecting to the protective relay. Fig. 21 provides the Security-Enhanced Linux policy statements to ensure that only the *trusted_web_t* and *scada_protection_t* domains are able to access device files labeled with the *scada_protection_t* label.

```
# relay collection has read/write access to the device file.
allow scada_protection_t scada_protection_tty_t
        :chr_file { read write ioctl };

# trusted web has relabel access to the device file.
allow trusted_web_t scada_protection_tty_t
        :chr_file { getattr relabelfrom relabelto };

# prohibit all other access to the device file.
neverallow ~{ scada_protection_t trusted_web_t }
        scada_protection_tty_t:chr_file *;
neverallow scada_protection_t scada_protection_tty_t
        :chr_file ~{ read write ioctl };
neverallow trusted_web_t scada_protection_tty_t
        :chr_file ~{ getattr relabelfrom relabelto };
```

Fig. 21. SCADA Protection Domain—Device File Policy Statements

Fig. 22 provides the Security-Enhanced Linux policy statements to ensure the *scada_protection_t* domain only has privileges to access IP addresses using the SCADA network interface.

```
# event report collection needs read/write permissions
# to the SCADA network
allow scada_protection_t trusted_t
        :netif { tcp_recv tcp_send };

# prohibit access to all other network interfaces.
neverallow scada_protection_t ~trusted_t
        :netif *;
```

Fig. 22. SCADA Protection Domain Only Permitted to Communicate Over a Trusted Network Interface

Data collected from the relay data collection application is stored in an SQLite database in the */database/protection* directory. Fig. 23 provides the Security-Enhanced Linux policy statements to ensure only the *scada_protection_t* domain has privileges to create, remove, read, and write files in the */database/protection* directory.

```
# relay collection has create/read/write/delete permissions
allow scada_protection_t scada_protection_t
        :file { create read getattr write unlink lock };

# web protection has read permissions
allow web_protection_t scada_protection_t
        :file { read getattr lock };

# init domain requires read permissions
allow init_t scada_protection_t:file read;

# prohibit all other access to the SCADA data
neverallow ~{ scada_protection_t web_protection_t init_t }
        scada_protection_t:file *;
neverallow scada_protection_t scada_protection_t
        :file ~{ create read getattr write unlink lock };
neverallow web_protection_t scada_protection_t
        :file ~{ read getattr lock };
neverallow init_t scada_protection_t:file ~read;
```

Fig. 23. SCADA Protection Domain—Database File Permissions

Data collected from the event reports collection application is stored in the */var/spool/event_reports* directory. The event reports are labeled with the *event_reports_t* label to experiment with assigning different permissions to data from the same SCADA device. Fig. 24 provides the Security-Enhanced Linux policy statements to ensure the *scada_protection_t* domain has privileges to create and write but not delete or read files with the *event_reports_t* label.

```
# event reports can add files to the directory.
allow scada_protection_t event_reports_t
        :dir { search write add_name };

# event reports can create and write files.
allow scada_protection_t event_reports_t
        :file { write create getattr };

# prohibit all access to event reports except to the
# scada_protection_t and web_protection_t domains.
neverallow ~{ scada_protection_t web_protection_t }
        event_reports_t:file *;

# verify scada_protection_t permissions to event_report_t
# files.
neverallow scada_protection_t event_reports_t
        :file ~{ write create getattr };
```

Fig. 24. SCADA Protection Domain—Event Reports Permissions

The Security-Enhanced Linux policy statements provided in this section show how the *scada_protection_t* domain participates in the one-way communications design of the Security-Enhanced Linux SCADA proxy. Fig. 21 shows the *scada_protection_t* domain is the only domain with privileges to communicate with device files with the *scada_protection_tty_t* label. Fig. 22 shows the *scada_protection_t* domain as the only domain with privileges to communicate with user-specified IP addresses on the SCADA network. Fig. 23 and Fig. 24 show the *scada_protection_t* domain as the only domain with privileges to update the SQLite database in the */database/protection* directory and the event reports in the */var/spool/event_reports* directory. Fig. 25 provides the final policy statement to ensure all communications paths are prohibited between the application domains only connected by white space in Fig. 14.

```
neverallow { untrusted_web_t web_meter_t scada_meter_t }
        { scada_protection_t scada_protection_exec_t
        scada_protection_tty_t }:
        { blk_file chr_file dir fd fifo_file file filesystem
        lnk_file sock_file association key_socket netif
        netlink_audit_socket netlink_dnrt_socket
        netlink_ip6fw_socket netlink_kobject_uevent_socket
        netlink_nflog_socket netlink_route_socket sem shm
        netlink_selinux_socket netlink_socket msg msgq
        netlink_tcpdiag_socket netlink_xfrm_socket
        node packet_socket rawip_socket socket tcp_socket
        udp_socket unix_dgram_socket unix_stream_socket ipc
        capability passwd pax process security system } *;
```

Fig. 25. SCADA Protection Domain—White Space Verification

C. Untrusted Web Domain

The SCADA meter and SCADA protection domains are responsible for collecting and storing SCADA data on the Security-Enhanced Linux SCADA proxy. The *untrusted_web_t* domain is responsible for providing these data to authorized users on the corporate IT network. The untrusted web server executes common gateway interface (CGI) applications to access the SCADA data and presents these SCADA data to the authorized user. The CGI applications are marked as domain entry point executables so they will perform a default domain transition into the appropriate domain, thus having read access to the required SCADA data.

The Security-Enhanced Linux SCADA proxy uses three application domains to provide these data to corporate IT users as an experiment to explore the different levels of protection Security-Enhanced Linux can offer. This area still needs continued research, but the current level of protection is:

- Compromised *web_meter_t* domain – the intruder has full read-only access to all stored SCADA data from the meter.
- Compromised *web_protection_t* domain – the intruder has full read-only access to all stored SCADA data from the protective relay.
- Compromised *untrusted_web_t* domain – the intruder does not have direct access to the stored SCADA data. The intruder does have permission to launch the CGI applications and read stored SCADA data from the meter and protective relay.

Fig. 26 provides the Security-Enhanced Linux security policy to ensure the untrusted web server can only communicate with the untrusted network interface on the HTTPS port (443/tcp).

```
allow untrusted_web_t untrusted_t
        :netif tcp_recv;
allow untrusted_web_t https_port_t
        :tcp_socket name_bind;

neverallow untrusted_web_t ~untrusted_t
        :netif *;
neverallow untrusted_web_t ~https_port_t
        :tcp_socket name_bind;
```

Fig. 26. Untrusted Web Domain Network Interface Security Policy

Fig. 27 provides the Security-Enhanced Linux security policy to ensure the *untrusted_web_t* domain is only permitted to transition into the *web_meter_t* and *web_protection_t* domains.

```
neverallow untrusted_web_t ~{ web_meter_t web_protection_t }
        :process transition;
```

Fig. 27. Untrusted Web Domain—Domain Transitions

The Security-Enhanced Linux policy statements provided in this section show how the *untrusted_web_t* domain participates in the one-way communications design of the Security-Enhanced Linux SCADA proxy. Fig. 26 shows that the *untrusted_web_t* domain only has permission to bind to port 443/tcp on the corporate IT network. Fig. 27 shows that the *untrusted_web_t* domain only has permission to transition into the *web_meter_t* and *web_protection_t* domains. Fig. 28 provides the final policy statement to ensure all communications paths are prohibited between application domains only connected by white space in Fig. 14.

```
neverallow { scada_meter_t scada_protection_t }
        { untrusted_web_t untrusted_web_exec_t }:
        { blk_file chr_file dir fd fifo_file file filesystem
        lnk_file sock_file association key_socket netif
        netlink_audit_socket netlink_dnrt_socket
        netlink_ip6fw_socket netlink_kobject_uevent_socket
        netlink_nflog_socket netlink_route_socket sem shm
        netlink_selinux_socket netlink_socket msg msgq
        netlink_tcpdiag_socket netlink_xfrm_socket
        node packet_socket rawip_socket socket tcp_socket
        udp_socket unix_dgram_socket unix_stream_socket ipc
        capability passwd pax process security system } *;
```

Fig. 28. Untrusted Web Domain—White Space Verification

The *untrusted_web_t* domain is the most susceptible to intruder attacks because it is the only port listening on the *untrusted_t* network interface. The Security-Enhanced Linux policy statements provided in Fig. 29 ensure the enhancements and improvements to the Security-Enhanced Linux SCADA proxy do not reduce the security offered by the original Security-Enhanced Linux SCADA proxy proof-of-concept. These policy statements ensure processes running in the *untrusted_web_t* domain are unable to access any serial ports and are unable to create, delete, or modify any files on the Security-Enhanced Linux SCADA proxy.

```
# prohibit create, delete, or write access to all files.
neverallow untrusted_web_t *
        :file { create unlink write };

# prohibit access to all serial ports.
neverallow untrusted_web_t ~{ urandom_device_t null_device_t }
        :chr_file *;
```

Fig. 29. Untrusted Web Domain File and Serial Port Security Policy

D. Untrusted Web Meter Domain

The *web_meter_t* domain is responsible for presenting the SCADA data from the revenue meter to the business group in a useful format. As discussed in [7], this proof-of-concept provides the business group with three methods for accessing the SCADA data from the revenue meter: a front-panel display, a graph showing power usage, and a graph showing the power factor. Fig. 30 provides a detailed look at how the *web_meter_t* domain participates in the one-way communications path through the SCADA data proxy domains.

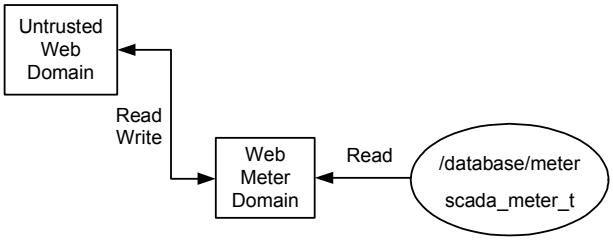

Fig. 30. Web Meter Domain Permissions

Fig. 31 provides the Security-Enhanced Linux policy statements to ensure that only the *web_meter_t* domain has read-only permission to the SCADA data stored in the SQLite database in the */database/meter* directory.

```
allow web_meter_t scada_meter_t
        :file { read getattr lock };

neverallow web_meter_t scada_meter_t
        :file ~{ read getattr lock };
```

Fig. 31. Web Meter Domain—Database File Permissions

Fig. 32 provides the Security-Enhanced Linux policy statements showing the read/write relationship between the *web_meter_t* and *untrusted_web_t* domains.

```
allow web_meter_t untrusted_web_t
        :tcp_socket { read write };

neverallow ~{ untrusted_web_t web_meter_t web_protection_t }
        untrusted_web_t:tcp_socket { read write };
neverallow web_meter_t untrusted_web_t
        :tcp_socket ~{ read write };
```

Fig. 32. Web Meter Has Read/Write Permissions to Untrusted Web Domain

The Security-Enhanced Linux policy statements provided in this section show how the *web_meter_t* domain participates in the one-way communications design of the Security-Enhanced Linux SCADA proxy. As shown previously in Fig. 18, the *web_meter_t* domain only has read permission to the SCADA data collected from the revenue meter. Fig. 33 provides the final policy statement to ensure all

communications paths are prohibited between application domains only connected by white space in Fig. 14.

```
neverallow { web_protection_t scada_protection_t }
        { web_meter_t web_meter_exec_t }:
        { blk_file chr_file dir fd fifo_file file filesystem
        lnk_file sock_file association key_socket netif
        netlink_audit_socket netlink_dnrt_socket
        netlink_ip6fw_socket netlink_kobject_uevent_socket
        netlink_nflog_socket netlink_route_socket sem shm
        netlink_selinux_socket netlink_socket msg msgq
        netlink_tcpdiag_socket netlink_xfrm_socket
        node packet_socket rawip_socket socket tcp_socket
        udp_socket unix_dgram_socket unix_stream_socket ipc
        capability passwd pax process security system } *;
```

Fig. 33. Web Meter Domain—White Space Verification

The *web_meter_t* domain is potentially accessible to an intruder from the corporate IT network. The Security-Enhanced Linux policy statements in Fig. 34 ensure the *web_meter_t* domain is prohibited from accessing the network interfaces, accessing the serial ports, and modifying files on the Security-Enhanced Linux SCADA proxy.

```
# prohibit access to all network interfaces.
neverallow web_meter_t *:netif *;

# prohibit access to all serial ports.
neverallow web_meter_t ~null_device_t
        :chr_file *;

# prohibit create/write/delete of all files.
neverallow web_meter_t *
        :file { create unlink write };
```

Fig. 34. Web Meter Cannot Access Network Interface and Serial Ports or Modify Files

E. Untrusted Web Protection Domain

The *web_protection_t* domain is responsible for presenting the SCADA data from the protective relay to protection engineers in a useful format. As discussed in [7], this proof-of-concept provides the protection engineers with two methods for accessing the SCADA data from the protective relay: a one-line diagram of the test environment and an event report viewer. Fig. 35 provides a detailed look at how the *web_protection_t* domain participates in the one-way communications path through the SCADA data proxy domains.

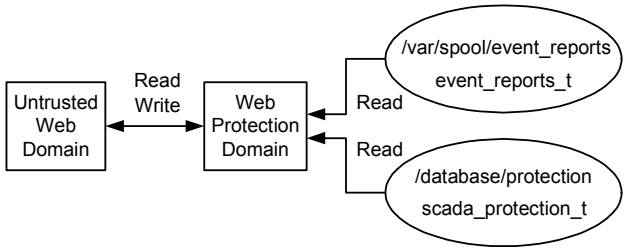

Fig. 35. Web Protection Domain Permissions

Fig. 36 provides the Security-Enhanced Linux policy statements permitting the *web_protection_t* domain read-only access to protective relay data stored in the SQLite database in the */database/protection* directory.

```
allow web_protection_t scada_protection_t
        :file { read getattr lock };

neverallow web_protection_t scada_protection_t
        :file ~{ read getattr lock };
```

Fig. 36. Web Protection Domain—Database File Permissions

Fig. 37 provides the Security-Enhanced Linux policy statements permitting the *web_protection_t* domain read-only access to the event reports stored on the Security-Enhanced Linux SCADA proxy.

```
allow web_protection_t event_reports_t
        :file { read getattr  };

neverallow web_protection_t event_reports_t
        :file ~{ read getattr };
```

Fig. 37. Web Protection Domain—Event Reports Permissions

The Security-Enhanced Linux policy statements provided in this section show how the *web_protection_t* domain participates in the one-way communications design of the Security-Enhanced Linux SCADA proxy. Fig. 36 and Fig. 37 show that only the *web_protection_t* domain has read-only access to the protective relay data. Fig. 38 provides the final policy statement to ensure all communications paths are prohibited between application domains only connected by white space in Fig. 14.

```
neverallow { web_meter_t scada_meter_t }
        { web_protection_t web_protection_exec_t }:
        { blk_file chr_file dir fd fifo_file file filesystem
        lnk_file sock_file association key_socket netif
        netlink_audit_socket netlink_dnrt_socket
        netlink_ip6fw_socket netlink_kobject_uevent_socket
        netlink_nflog_socket netlink_route_socket sem shm
        netlink_selinux_socket netlink_socket msg msgq
        netlink_tcpdiag_socket netlink_xfrm_socket
        node packet_socket rawip_socket socket tcp_socket
        udp_socket unix_dgram_socket unix_stream_socket ipc
        capability passwd pax process security system } *;
```

Fig. 38. Web Protection Domain—White Space Verification

The *web_protection_t* domain is potentially accessible to an intruder from the corporate IT network. The Security-Enhanced Linux policy statements in Fig. 39 ensure the *web_protection_t* domain is prohibited from accessing the network interfaces, accessing the serial ports, and modifying files on the Security-Enhanced Linux SCADA proxy.

```
# prohibit access to all network interfaces.
neverallow web_protection_t *:netif *;

# prohibit access to all serial ports.
neverallow web_protection_t ~null_device_t
        :chr_file *;

# prohibit create/write/delete of all files.
neverallow web_protection_t *
        :file { create unlink write };
```

Fig. 39. Web Protection Domain Cannot Access Network Interface and Serial Ports or Modify Files

IV. Physical Contact Inputs and Outputs

The Security-Enhanced Linux security policy provides support for run-time conditional (i.e., Boolean) variables. This allows the security policy to dynamically switch between pre-defined behaviors, depending upon the status of the conditional variable. The platform the Security-Enhanced Linux SCADA proxy is running on supports a single physical contact input and a single physical contact output (without the addition of an expansion board). The use of a single physical contact of each type is not a restriction of the Security-Enhanced Linux policy language, in fact, the ideas presented in this paper scale to multiple physical contact inputs and multiple physical contact outputs.

A. Physical Contact Input

The high-level integration goal for the physical contact input is to only permit changes to the electronic device settings when the physical contact input is asserted and to prohibit all changes to the electronic device settings when the physical contact input is deasserted. To realize this integration goal, a custom Linux driver was added to the Linux kernel to periodically scan the status of the physical contact input. This Linux kernel driver detects rising and falling edges to appropriately set the *update_settings* conditional variable in the Security-Enhanced Linux security policy.

The Security-Enhanced Linux security policy uses the *update_settings* conditional variable to dynamically grant or revoke privileges from the *trusted_web_t* domain. The *trusted_web_t* domain gains the following privileges when the physical contact input is asserted:

- Write permissions to files with the *trusted_web_settings_t* label.[9]
- Relabeling permissions on device files associated with serial ports.[10]
- Reboot privileges.[11]

When the physical contact input is deasserted, the above privileges are revoked from the *trusted_web_t* domain.

Fig. 40 provides the relevant Security-Enhanced Linux policy statements to grant and revoke the appropriate privileges from the *trusted_web_t* domain.

```
if(update_settings) {
      ## relabel device files.
      allow trusted_web_t security_t:filesystem getattr;
      allow { scada_meter_tty_t scada_protection_tty_t } fs_t
           :filesystem associate;

      allow trusted_web_t security_t:dir search;
      allow trusted_web_t security_t:file { read write };
      allow trusted_web_t { scada_meter_tty_t
           scada_protection_tty_t tty_device_t }
           :chr_file { getattr relabelto relabelfrom };
      allow trusted_web_t security_t:security check_context;

      ## Setting Files
      allow trusted_web_t trusted_web_settings_t
           :file { read getattr write };

      ## Reboot
      allow init_t init_t:capability sys_boot;
}
```

Fig. 40. Conditional Security-Enhanced Linux Policy Statements

B. Physical Contact Output

The high-level integration goal for the physical contact output is to provide notification when Security-Enhanced Linux policy alarms are generated. In addition to sending Security-Enhanced Linux policy alarm messages to the central syslog server, the custom Linux kernel driver also asserts the physical contact output. This output can be used to provide a visible alarm such as driving an LED or controlling an alarm icon on an HMI (human-machine interface) screen.

V. Device Settings

The final goal for this paper is to extend the functionality of the original Security-Enhanced Linux SCADA proxy to allow users to make changes to the electronic device settings. This proof-of-concept Security-Enhanced Linux SCADA proxy permits users to modify the following device settings using the trusted web server: network interface settings, syslog server settings, revenue meter communications settings, and protective relay communications settings.

The following four sections describe in detail how the Security-Enhanced Linux SCADA proxy implements the user-updateable settings. To simplify the settings description, assume the physical contact input is asserted.

A. Network Interface Settings

The trusted web server allows users to configure the IP address and the subnet mask for both the trusted and untrusted network interfaces.[12] The Security-Enhanced Linux SCADA proxy configures the network interface in the */etc/inittab* file.[13]

[9] Files with the *trusted_web_settings_t* label are configuration files the trusted web server is permitted to update on settings changes.

[10] To ensure the data collection domains remain isolated, the *scada_meter_t* and *scada_protection_t* domains require the appropriate label on the device file. The trusted web server must relabel these device files with the appropriate label when specifying which serial port the SCADA device is connected to.

[11] After changes to the settings on the device have been made, the system requires a reboot before the settings changes take effect. Online device changes are possible but were not implemented in this proof-of-concept.

[12] The trusted network interface is labeled with the *trusted_t* label, while the untrusted network is labeled with the *untrusted_t* label.

[13] The init program reads the */etc/inittab* file on system startup to determine which applications need to be started.

Although granting the *trusted_web_t* domain write privileges to the */etc/inittab* file would provide user-updateable settings, it would also expose the entire */etc/inittab* file to an intruder attacking from the SCADA network. Because the Security-Enhanced Linux SCADA proxy was designed to protect itself against both corporate IT and SCADA intruders, an alternative method for updating the */etc/inittab* was used.

This alternative method has the *trusted_web_t* domain write the IP address and subnet information for each network interface to *user-configuration-files*. The nonuser-updateable */etc/inittab* entries are stored in */etc/inittab fragments*. The custom *buildinit* shutdown application regenerates the */etc/inittab* file from the */etc/inittab fragments* and the *user-configuration-files*. The custom *buildinit* shutdown application performs input validation on the user-defined data before generating proper */etc/inittab* entries. This input validation is important to prevent an intruder from making unauthorized modifications to the */etc/inittab* file.[14]

B. Syslog Server Settings

The trusted web server allows users to configure the IP address for the syslog server. The syslog server application is also launched from the */etc/inittab* file. The syslog server IP address is updated in the */etc/inittab* file using the same methodology described in the previous section on network interface settings. In addition to updating the */etc/inittab* file, the syslog server IP address must also be updated in the firewall rules and the Security-Enhanced Linux security policy.

The Linux operating system provides a stateful, packet-filtering firewall built directly into the Linux kernel. The firewall rules are stored in the */etc/rc.firewall* file and loaded into the Linux kernel during system boot.[15] Similar to how the */etc/inittab* is regenerated, a custom *buildfirewall* shutdown application regenerates the */etc/rc.firewall* file. To simplify and prevent configuration errors, the custom *buildinit* and custom *buildfirewall* applications use the same syslog *user-configuration-file*.

Following the security-in-depth principle, the Security-Enhanced Linux security policy also ensures syslog messages are only sent to the specified syslog server. The Security-Enhanced Linux security policy offers greater protection by ensuring only the *syslog_t* domain is permitted to send syslog messages to the syslog server.

To prevent a covert (unauthorized) communications channel through the Security-Enhanced Linux SCADA proxy, the syslog server must exist on the corporate IT network. Fig. 41 provides the Security-Enhanced Linux policy statements to ensure that only the *syslog_t* domain can send syslog messages to the syslog server.

[14] There is a very small chance an intruder from the SCADA network could still make unauthorized modifications to the */etc/inittab* file. The *trusted_web_t* domain transitions into the *init_t* domain (via the *reboot.cgi* application), and the *init_t* domain has privileges to write to the */etc/inittab* file. Experimentations to separate the reboot command from the *init_t* domain were not completed in time for this paper.

[15] The firewall rules are loaded into the Linux kernel before the network interfaces are configured.

```
allow syslog_t untrusted_t:netif udp_send;
allow syslog_t syslog_node_t:node udp_send;

neverallow syslog_t ~untrusted_t:netif *;
neverallow syslog_t untrusted_t:netif ~udp_send;
neverallow syslog_t ~syslog_node_t:node udp_send;
```

Fig. 41. Untrusted Network Syslog Security Policy Statements

Fig. 41 introduces the *syslog_node_t* type representing the syslog server. Fig. 42 provides the Security-Enhanced Linux policy statement to associate the user-specified IP address with the *syslog_node_t* type.

```
nodecon <IPAddress> 255.255.255.255
                system_u:object_r:syslog_node_t
```

Fig. 42. *Syslog_node_t* Security-Enhanced Linux Policy Definition

Following the methodology of the custom *buildinit* and *buildfirewall* applications, the *buildpolicy* application reads, validates, and generates the nodecon Security-Enhanced Linux policy statement with the proper <IPAddress> based on the format in Fig. 42. After the policy has been generated, the *buildpolicy* application then compiles the policy into the binary form required by the Linux kernel.[16]

C. Revenue Meter Communications Settings

For this proof-of-concept, the only user-configurable setting is which serial port the revenue meter is connected to. Additional user-configurable settings were not added because they did not add anything new to this paper.

As discussed above, the *trusted_web_t* domain updates a meter *user-configuration-file*. The meter *user-configuration-file* contains the name of the device file the meter data collection process will use to connect to the revenue meter. In addition to specifying which device file to use, the device file must also have the *scada_meter_t* label.

To prevent the *trusted_web_t* domain from having permission to relabel any file, the *trusted_web_t* domain is only permitted to relabel device files with the *tty_device_t* label to either the *scada_meter_t* or *scada_protection_t* labels. With this restriction in place, the *trusted_web_t* domain must relabel both the previous device file with the *tty_device_t* label and the new device file with the *scada_meter_tty_t* label.

D. Protective Relay Communications Settings

The *scada_protection_t* domain also provides a user-configurable setting to identify which serial port the protective relay is connected to. The methodology for updating the relay *user-configuration-file* and the meter *user-configuration-file* is identical.

In addition to collecting SCADA data over a serial connection, the *scada_protection_t* domain also collects the event reports using a network connection. The relay *user-configuration-file* contains both the device file and the IP address of the protective relay. Similar to the syslog server

[16] The new Security-Enhanced Linux security policy will take effect after the reboot. If online changes are desired, Security-Enhanced Linux can also be configured to reload the policy without rebooting.

settings in Section B above, changing the IP address of the protective relay requires updates to both the Linux firewall and Security-Enhanced Linux security policy. The *buildfirewall* and *buildpolicy* applications discussed in Section B also update the Linux firewall and Security-Enhanced Linux security policy with the protective relay IP address information.

VI. Conclusion

The first objective of this paper was to expose how the Security-Enhanced Linux security policy used on the Security-Enhanced Linux SCADA proxy enforces a one-way communications path of SCADA data to authorized users on the corporate IT network. Through the careful management of file and directory privileges, the Security-Enhanced Linux policy statements provided in this paper support the one-way communications diagram illustrated in Fig. 14.

The second objective was to extend the research into the Security-Enhanced Linux SCADA proxy by integrating device settings and physical contact inputs and outputs into the Security-Enhanced Linux security policy and standard device operation. Although the device settings are incomplete, a variety of settings were implemented to show how device settings could be implemented with minimal impact to the security goals and objectives of the Security-Enhanced Linux SCADA proxy. The additional security protection from using the physical contact input to dynamically switch between the predefined policy behaviors (has privileges to write settings or does not have privileges to write settings) worked very well and would be interesting to study further.

Although this paper takes a detailed look into the Security-Enhanced Linux security policy, most of what is described in this paper is not end-user visible. The only end-user-visible portion of the Security-Enhanced Linux security policy is the generated syslog messages when the Security-Enhanced Linux SCADA policy statements are violated.

VII. References

[1] P. Oman, E. O. Schweitzer III, and D. Frincke, "Concerns About Intrusions Into Remotely Accessible Substation Controllers and SCADA Systems," proceedings of the 27th Annual Western Protective Relay Conference, Spokane, WA, October 2000.

[2] C. Leonard, "Taum Sauk Reservoir Fails," *seMissourian.com News*, December 14, 2005. Available: http://www.semissourian.com/story /1131377.html

[3] S. Boyer, *SCADA: Supervisory Control and Data Acquisition*, 2nd ed., North Carolina: ISA, 1999.

[4] G. Clarke and D. Reynders, *Practical Modern SCADA Protocols: DNP3, IEC 60870.5 and Related Systems*, Massachusetts: Newnes, 2004.

[5] F. Mayer, K. MacMillan, and D. Caplan, *SELinux by Example: Using Security Enhanced Linux*, Prentice Hall, 2006.

[6] R. Bradetich and P. Oman, "Connecting SCADA Systems to Corporate IT Networks Using Security-Enhanced Linux," proceedings of the 34th Annual Western Protective Relay Conference, Spokane, WA, October 2007.

[7] R. Bradetich, "Using SE-Linux Object Type Enforcement Domains to Logically Isolate SCADA Networks," Master's thesis, University of Idaho, Moscow, ID, December 2007.

VIII. Biographies

Ryan Bradetich received his BSCS in 1997 and his MSCS in 2007 from the University of Idaho. He is a lead database developer at Schweitzer Engineering Laboratories, Inc. (SEL) in Pullman, WA. Ryan is currently working on automation products used in electric utility substations. Prior to joining SEL, he worked at Hewlett-Packard on the security team responsible for auditing and reporting the security status for approximately 20,000 UNIX and Windows® systems.

Dr. Paul W. Oman is a Professor of Computer Science at the University of Idaho. He is currently working on secure communications and critical infrastructure protection with grants from NSF, NIATT, and DARPA. From 2000 to 2002, he served as a senior research engineer at Schweitzer Engineering Laboratories, Inc. (SEL), specializing in digital equipment for electric power system protection. Before joining SEL, he was Chair of the CS Department at the University of Idaho and held the distinction of Hewlett-Packard Engineering Chair for a period of seven years. He is a Senior Member of the IEEE.