Identifying Attacks on Control Systems by Scripting Event Aggregation and Correlation

Ron Gula
Tenable Network Security, Inc.
8830 Stanford Blvd., Suite 312, Columbia, MD 21405
rgula@tenablesecurity.com

Abstract: Potential security events are logged in a variety of systems and locations such as SCADA application logs, firewall logs, IDS logs, infrastructure system logs and operating system logs. Solutions exist from Tenable Network Security and other vendors to correlate security events from multiple sources to detect attacks, but the meta events and correlation methods must be defined and developed. This paper discusses and provides TASL scripts to detect three different attack scenarios: SCADA server compromises, malicious SCADA server failover and DNP3 unsolicited response flood. Ideas for the detection through event correlation are presented for four other control system meta events.

Keywords: Event Correlation, Log Analysis, SCADA, SEM

1 Introduction

The term "SCADA" is an acronym which stands for "Supervisory Control And Data Acquisition". It represents a family of protocols which can be used to monitor and manage a variety of machinery and equipment involved with many activities including:

- Power generation and distribution
- Manufacturing processes
- Large-scale chemical processes
- Transportation of materials

SCADA systems are used for many different types of processes which require monitoring, reporting or control from a computer system. SCADA devices are managed and report information to control centers through a variety of protocols including DNP3, ICCP and Modbus protocols. SCADA systems are used for control of systems requiring both human and automated interaction.

Since SCADA networks control very critical processes, any discussion of hackers taking them over or a worm causing an outage quickly escalate into "Hollywood" or "worse-case" scenarios. Too often we see examples in the movies (such as Trinity in "Matrix 2" breaking into a power system) or books about how easy it is for these networks to be compromised.

The reality is that the actual SCADA protocols are subject to the same sorts of attack techniques that email, web servers and file transfer protocols are subject to. These include denial of service, buffer overflows and so on. Any vendor who sells a SCADA device may have made the same sort of programming errors that Cisco, Microsoft of any number of other vendors have made.

To make matters worse, the SCADA devices themselves live on a network which may have other types of traditional vulnerabilities. For example, the servers that run a SCADA protocol may also be running an unauthorized web server or have an un-patched kernel. Any vulnerability in an underlying device on a SCADA network may ultimately result in a potential to control or disable the entire SCADA network.

This paper will discuss three "attack" scenarios which involve SCADA networks. Methods for detecting each of the attack scenarios will be presented which make use of the Log Correlation Engine (LCE) product from Tenable Network Security. Each example has a variety of log analysis and algorithms applied. The algorithms discussed are written in the Tenable Application Scripting Language (TASL). The intent of the exercises is to both understand the issues surrounding monitoring of SCADA networks, as well as implementation issues typically encountered.

2 Tenable Log Analysis and Correlation Technology

Tenable Network Security offers a variety of log analysis solutions centered around the Log Correlation Engine. This product can aggregate real time log data that exists in SYSLOG and flat log files into a normalized data set. Log parsing libraries for most firewalls, operating systems, netflow and intrusion detection devices are included with the LCE.

As logs are aggregated to the LCE, they are normalized and correlated for statistical anomalies and run through a set of correlation scripts known as TASL scripts. This paper will consider four different TASL scripts specific to SCADA logs and the three attack scenarios.

All log parsing is accomplished with libraries referred to as "PRM" files. Each library has a combination of strict pattern and regular expression rules for high speed identification of log types. For logs sent to the LCE, the normalization process will result in a unique ID being set for the log type as well as a normalized name. For example, a log generated by a Solaris server for a console login failure might have an event ID of 25373 and be named "Console-Login_Failure".

2.1 Event Correlation with TASL Scripts

The TASL scripting language is comprised of several components. First, each script "subscribes" to just the events they need. The subscription can be based on IP addresses, ports or specific types of events. Second, there is some logic implemented by the TASL. This logic can be very simple "if A then B" type of actions, or very complex algorithms. Lastly, the TASL script can generate a new alert. This is a log like any other log, so it needs a PRM library associated with it. These new logs can also be fed into other TASL scripts for very complex interactions.

2.2 Existing TASL Scripts

Tenable keeps an archive of all TASL scripts available on the web and also makes several posts a month regarding TASL writing and log correlation at our blog. Both can be accessed at the following URLs:

- http://cgi.tenablesecurity.com/tasl.html
- http://bog.tenablesecurity.com

3 Scenario #1 - Detecting SCADA Server Compromises

This scenario is for large scale networks which operate SCADA systems. The problem we are trying to solve is to detect when a "SCADA system" (specifically using that term loosely) has become compromised. If there are intrusion detection or network anomaly logs, there will likely be too many to process manually or make sense do to sheer volume of traffic.

For this scenario we will leverage the use of an "asset list" in the LCE. An asset list is simply a list of IP addresses or networks which have a certain quality. They could be web servers, things that speak DNP3 or IP addresses which resolve DNS names that contain the string "corporate". We also want to consider not only any type of device which speaks a SCADA protocol like DNP3 or ICCP, but also supporting devices such as the routers, switches, firewalls, desktops and servers necessary for operation. From a NERC perspective, this could be the list of all IP addresses in the Critical Cyber Assets.

Although not the focus of this paper, Tenable does have solutions which actively and passively identify devices which speak SCADA protocols. They can also make use of Active Directory and DNS names as well as IP network addressing to dynamically create the list of SCADA devices.

3.1 Sources of Log Data

For this scenario, we are also assuming a large range of available log sources. These can include much of the following:

- Netflow
- Windows OS logs
- UNIX OS logs

- Router logs
- Intrusion Detection logs

Notice that of yet, we have not considered logs specifically from a SCADA device.

3.2 Algorithmic Approach

If a user were simply to correlate any of these log types with our "SCADA" asset list, they could accomplish that using the asset list as a filter. Below is a screen shot of Tenable's Security Center which is used to perform queries into vulnerability and log data.

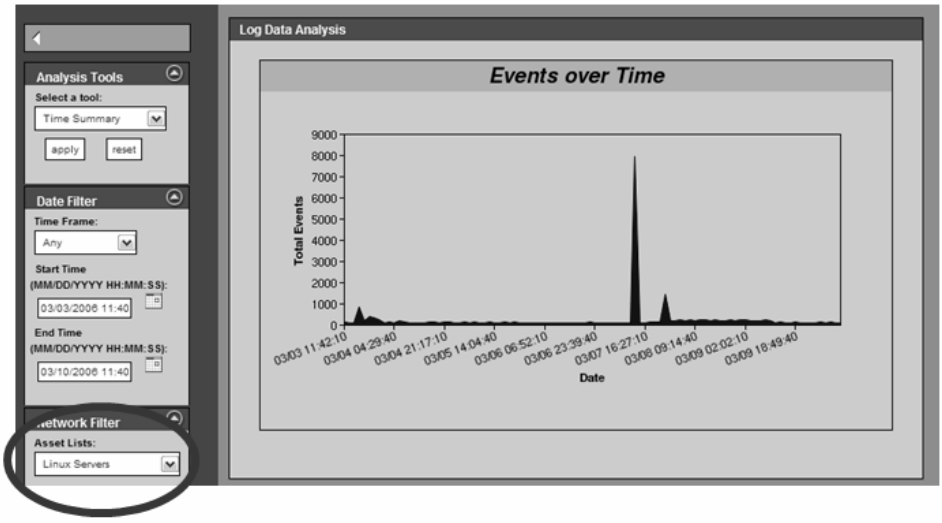

Figure 1 – Events Filtered by Asset Lists

In this screen, a user selected an asset list named "Linux Servers" and then chose to chart log activity over time.

This approach is useful for analysis and reporting, but not for general high-quality alerting. A large SCADA network will have many types of logs from SCADA devices and we wish to look for high-quality alerts. Unlike academic, government or commercial networks, one would hope that there would not be a large number of changes, Internet browsing or network chat on a SCADA network. Because of this, we can leverage some of Tenable's existing TASL scripts including:

> ***botnet_with_scan.tasl*** – correlates various IDS alerts for Botnet activity with active exploit scanning
>
> ***ids_event_followed_by_change.tasl*** – correlates various IDS alerts with system logs which indicate a change in configuration or activity

nids_compromise_correlate.tasl – correlates when two different network IDS types (Snort, Tipping Point, ISS, Cisco, .etc) agree there has been a serious attack

nids_attack_bounce.tasl – uses IDS alerts to correlate when a target has been attacked and then begins attacking other targets

successful_login_after_multiple_failures.tasl – looks for succseful brute force password guessing across UNIX and Windows logs

These TASL scripts are available online at the http://cgi.tenablesecurity.com/tasl.html web site. They provide a much higher sense of correlation for IDS event analysis. However, they are still dependent upon the IDS events for the majority of their content. Tenable has also developed a separate class of "policy" TASL scripts which can be used to look for changes. These are listed here:

compliance_check.tasl – performs a variety of ISO 1779, PCI and SOX alerting on a variety of UNIX and Windows log files

invalid_user_login.tasl – gathers user names for Windows systems that have been disabled or deactivated

detect_change.tasl – looks for network, device, system and user configuration changes

new_user.tasl – looks at UNIX and Windows user logs to automatically detect when a new user account has been accessed

The idea with these "policy" events is that we should not get a lot of them for a SCADA network. Of course new users are added over time and changes to systems are occurring, however, from an audit and trending point of view, these events are very useful.

We will now put these events all together with our "SCADA" asset list.

3.3 scada_abuse.tasl

Appendix C has a listing of the *scada_abuse.tasl* script.

This script assumes that an asset list named "SCADA.ip" is present. If the LCE is being managed by the Security Center, then a dynamic asset list can be created of SCADA devices from active scanning or passive network analysis. All of the IP addresses for the network are loaded into a hash table. This makes it very easy for a programmer to efficiently test if a string exists in a list later on.

The script then "subscribes" to the ID's of all the normalized logs for the above TASL scripts. There may be 100's or even 1000's of unique event ID's flowing through a Log Correlation Engine. This subscription method allows the LCE to operate with very high event insertion rates without the need to invoke each correlation rule. During this phase, our algorithm also marks each event as being a high quality IDS event or a change event. This is set in the *type[]* array.

The actual script starts at the *myCallBack()* function. This is the entry point that the LCE will call when a subscribed event occurs. The first thing our algorithm does is to make sure the source IP or destination IP of the event is part of our SCADA asset group. A quick call to the *ip_list[]* hash table returns a 1 if this is true.

If the script gets a 1, then a second lookup to the *type[]* array occurs which generates a SCADA message. The message contains information about a compromise or a configuration change on a SCADA device.

In plain English:

> For the list of things I know as SCADA devices, for all of the normalized logs, generate an alert for any significant device changes or compromise activity.

3.4 False Negative and False Positive Issues

So where can false positive events come from? Clearly if there are issues with the algorithms in the TASL scripts feeding into this script, we could get a false alert. We could also get a false alert if our list of SCADA devices wasn't correct and we included something like an Internet facing host running a web server.

Our false negative sources will likely come from attacks that the IDS events should have detected, but missed. Many of the TASLs feeding events into this script are based on IDS events and dependent on those. Also, we picked generic "OS" and "network" type logs to look for interesting traffic. If the data and logs used on a SCADA network are different than what we chose, we could be missing logs of interest.

4 Scenario #2a - Malicious SCADA Server Failover Correlation

One of the most critical things that can occur on a SCADA network is a failover of a controlling device. If an attacker is able to cause a device to fail over, they may be able to cause other devices to fail over in a cascade.

4.1 Sources of Log Data

We will consider fail-over logs from a Telvent OASyS DNA device. The PRM log file for just three events is located in Appendix A. There are several hundred log messages which can be generated by the OASyS device, but only a few are considered to illustrate our algorithm.

We will add these event IDs into the *scada_abuse.tasl* script and produce new logic for the *scada_abuse2.tasl* script in Appendix D. The goal of the script is to associate some sort of external stimulus and a failover event. Since this script is based on the last script, let's look at what has changed.

- We're now subscribing to event IDs 450, 451 and 452. These are the "fail over" events.
- Unless we get one of those events, we just log the IDS and policy change events.

- If we do get a "fail over" event we now ask the LCE to count the number of policy change and IDS events in the past 15 minutes. This lets the TASL script generate an alert.

In plain English:

> For the list of things I know as SCADA devices, for all of the normalized logs, generate an alert for any significant device changes or compromise activity.
>
> Also, if there are any failover events, see if they were preceded by any IDS events or compromise events in the last 15 minutes.

4.2 False Negative and False Positive Issues

For a false negative, we have less chance of "missing a log of interest" than with the original *scada_abuse.tasl* script because we are considering real logs. This focuses us in on one type of log, but there could be other logs we miss. Another factor in the false negative analysis is the time relation between the IDS event and the fail over event. Is 15 minutes too long, too short or just right?

False positive events are much more concerning. The concept of correlating failover events with high quality IDS events is very good. However, there is still a chance that a TASL script could be wrong and correlate incorrectly.

There is also a scenario where everything works as planned, but no one heeds the alert. If one assumes that we're not attacked successfully everyday, or even every week or month, then when it does occur, if the event isn't believed and ignored, then all is not worthwhile.

5 Scenario #2b – Malicious SCADA Server Failover Correlation

At a different organization, a SCADA security monitoring program only has access to network IDS events and system logs from the OASyS device. Can we do anything else with these logs without going through pre-correlation of IDS events? What would direct correlation of raw IDS events with SCADA "fail over" logs yield us?

Appendix E lists the third example (the *scada_abuse3.tasl* script) of a correlation script to look for SCADA devices with issues. The script is much different than the previous scripts:

- Subscribe to just a few "critical" events from various NIDS solutions
- Subscribe to the "fail over" log events
- Take no action when an IDS event occurs
- If a "fail over" event occurs, see if an IDS event occurred in the last 15 minutes and if so, throw a serious alert

5.1 False Negative and False Positive Issues

There are similar false negative issues as before. Are the IDS events there, do we have the proper time between these events and do we have the right log sources? Something we also did here is drop the concept of a "SCADA" asset. If we are just correlating with devices that log SCADA logs, they are surely SCADA devices right?

This algorithm is much more interesting from a false positive point of view. The reader may have trusted Tenable's TASL scripts to have a "lower" false positive rate than just dealing with raw IDS events. Since we are not using those correlation scripts in this script, what happens when we try to do this with IDS events that are more constant?

If you have worked with IDS events on a busy network, they occur often and roughly parallel network activity. Even though the *scada_abuse3.tasl* script only subscribes to critical or serious IDS events from a variety of vendors, we still have an algorithm which tries to correlate something that happens fairly often (an IDS event) with something that happens not that often (a controller fail over).

If an IDS event will have occurred sometime in the last 15 minutes against the controller, it will look like the remote attacker "caused" the fail over. If there have been no events in the last 15 minutes, then there obviously won't be a correlation.

6 Scenario #3 - DNP3 Unsolicited Response Flood

Our last scenario involves a simple denial of service (DoS) attack against a SCADA device which uses the DNP3 protocol. The DNP3 protocol is normally a poll and response mechanism. However, devices "in the field" can send unsolicited responses to indicate a problem or new data. This is an opportunity for an attacker to send data to a device and flood it with messages.

6.1 Sources of Log Data

We will consider two sources of log data. Tenable's Passive Vulnerability Scanner can alert when a new system, protocol, application or vulnerability is discovered, including devices which speak DNP3. The second source of data will be any log that works on port 20000, which is a default port for DNP3.

6.2 Detecting New SCADA Hosts

Tenable published a TASL script to complement the passive auditing of SCADA networks with the Passive Vulnerability Scanner (PVS). The script is named *new_scada.tasl* and is available at the http://cgi.tenablesecurity.com/tasl.html web site.

This script simply processes the real time "DNP3" detected events. It places all IP addresses which have been detected by the PVS into a hash table. Every time it gets an alert from a device that isn't in the hash table, it generates a "New SCADA Device" alert.

This is very useful for SCADA networks as devices which speak DNP3 aren't added very often. A new SCADA device event is a good way to audit changes to the network.

6.3 Flood Detection

Appendix F has a much less elegant way of looking for DNP3 unsolicited response floods. This script subscribes to any event that occurs on port 20000. This could be netflow, firewall logs, IDS events, PVS events and so on. For each event, the TASL counts all port 20000 events to the target host. If there are more than 5000, an alert is thrown and the source is suppressed so that we don't alert again on the 5001st packet, and again on the 5002nd packet.

This is clearly a threshold technique. Tenable generally tries to look for non-threshold algorithms in our TASL scripts because you need to tune them. In this case, there are two glaring variables to change. One is the time (five minutes), and the other is the count (5000 events).

The script is not as dumb as it looks though. If the DOS attack is distributed it, the *count()* API only considers total port 20000 events to the target. Maybe there were many targets sending the DOS events, each with a different source IP address. This script wouldn't identify it as a distributed attack, but it would still see the spike.

7 Ideas for More Correlation Algorithms

During the development of this paper, several different classes of attacks were discussed as potential examples. Although algorithms were not developed and tested, it made sense to include these here to provide the reader with examples.

7.1 Attackers Lying In Wait

A "Hollywood" nightmare is a set of sophisticated attackers who breach a SCADA network and maintain control over it until some point in the future where they start inflicting damage. It could be debated how these attackers get in and how they communicate with their infected devices. Having said that, several techniques could be developed to look for these situations:

- Correlating attacks with new services or processes running on a server
- Analyzing communications to look for backdoors
- Looking for behavior changes in SCADA devices
- Analyzing outbound proxy and firewall logs to see what is trying to connect out

7.2 Auditing User Administration Activity

For any type of control system, all users will have accounts which need to be added, activated and administered. A malicious attacker or insider will create user account with broad privileges or attempt to assume another person's role. Many of these basic

concepts are difficult to audit consistently on SCADA devices because there is no standard in the log formats of these events. Having said that, several scripts and log parsing routines could be written to perform:

- Auditing changes in a person's Area of Responsibility
- Correct parsing of critical system logs such as display changes
- Creation and de-activation of user accounts
- Attempts to use de-activated or un-authorized accounts
- Real user accounts logging into devices or locations they are unauthorized to do so
- Correlating these attempts with brute force password guessing and IDS attacks
- Performing time based analysis of user activity and alerting when they are out of "normal" time operations

7.3 Deeper SCADA System Event Correlation

Each of the following situations could easily be detected with a TASL script and the relevant system logs:

- For DNP3 floods, correlating a new SCADA device generating a flood
- Detecting when a "new" SCADA device is also launching port scans
- Looking for critical SCADA events (such as "fail over" events) across multiple devices within a short period. This may or may not be further correlated with broad attacks.

7.4 Detection of Internet Connections

A variety of network traffic, IDS events and even network logs can indicate that a device is connecting to the Internet or is attempting to connect to the Internet. On a network that is mostly "closed" being able to find these hosts quickly is vital. Several techniques can be used to look for these types of hosts:

- The *new_host_portscanning.tasl* is an existing script which identifies a new system which then starts to port scan other devices. It likely indicates that someone has plugged in an infected device which is beginning to probe.
- The existing *botnet.tasl* script can accept lists of networks known for sending SPAM email traffic. If you have a device on your network sending traffic to a known SPAM source, it may be infected with a rootkit or Trojan as part of a SPAM botnet.
- A script that learned your DNS servers by alerting on "new" servers could identify devices attempting to do name resolution to fixed IP addresses.

8 Conclusion

Analysis of SCADA logs to discover events for correlation can be a very complex undertaking. It requires knowledge of the underlying protocols, the logs generated by the SCADA devices and also understanding how our algorithms can fail and succeed. Often, the most difficult part of developing a SCADA correlation rule is to discover a need for a correlation that we didn't know about before.

About the Author – Ron Gula is President and Chief Technology Officer for Tenable Network Security, Inc. Tenable, located in Columbia, Md., develops enterprise security solutions that provide vulnerability management, intrusion detection, and security event notifications across entire organizations for effective network security management. Mr Gula was the original author of the Dragon IDS and CTO of Network Security Wizards which was acquired by Enterasys Networks. He began his career in information security while working at the National Security Agency conducting penetration tests of government networks and performing advanced vulnerability research. Mr. Gula has a BS from Clarkson University, an MSEE from the University of Southern Illinois, and was the recipient of the 2004 Techno Security Conference "Industry Professional of the Year" award.

Appendix A – scada_telvent_OASyS_DNA.prm

This PRM library will normalize logs from a Telvent OASyS DNA system. The event IDs in this file are consumed by several TASL scripts in the following appendixes.

```
id=450
name=OASyS DNA device log changing from STANDBY to FAIL
example=The REALTIME service on masterrts1 changed from STANDBY to
FAIL.
match=The REALTIME service on
match= changed from STANDBY to FAIL.
log=event:OASyS-STANDBY_to_FAIL  type:SCADA

NEXT

id=451
name=OASyS DNA device log initiation of automatic shutdown
example=Initiating automatic shutdown for service REALTIME on
computer masterrts1.  (Reason: 'Critical REALTIME process 'omnicomm'
dead on masterrts1.')
match=Initiating automatic shutdown for service REALTIME on computer
log=event:OASyS-Automatic_SHUTDOWN type:SCADA

NEXT

id=452
name=OASyS DNA device log failover succeeded
example=Failover of the REALTIME service on computer masterrts1
succeeded.  Proceeding with shutdown.
match=Failover of the REALTIME service on computer
log=event:OASyS-Failover_Suceeded type:SCADA
```

Appendix B – excerpt of lce_tasl.prm

These normalization rules are required for a functioning Log Correlation Engine to interpret the events generated by the *scada_abuse.tasl*, *scada_abuse2.tasl* and *dnp3_flood.tasl* scripts. The *lce_tasl.prm* library includes several 100 normalization rules and only the ones relevant to SCADA event processing are included here.

```
id=22030
name=The scada_abuse.tasl script observed a log monitored by policy.
example=SCADA-Policy_Event -
match=!TASL|
match=SCADA-Policy_Event -
log= event:SCADA-Policy_Event type:correlated

NEXT

id=22031
name=The scada_abuse.tasl script observed a log monitored by polcicy.
example=SCADA-Intrusion -
match=!TASL|
match=SCADA-Intrusion -
log= event:SCADA-Intrusion type:correlated

NEXT

id=22032
name=The scada_abuse.tasl script observed a system failover and
suspicious activity
example=SCADA-Suspicious_Failover -
match=!TASL|
match=SCADA-Suspicious_Failover -
log= event:SCADA-Suspicious_Failover type:correlated

NEXT

id=22033
name=The dnp3_flood.tasl script observed a more than 5000 DNP3 events
example=SCADA-DNP3_FLOOD –
match=!TASL|
match= SCADA-DNP3_FLOOD –
log= event:SCADA-DNP3_FLOOD type:correlated
```

Appendix C – scada_abuse.tasl

```
include ('functions.inc');
global_var ip_list;
ip_list = null;
ip_file = '/usr/thunder/admin/SCADA.ip';
# Subscribed events. Type of 1 is for IDS events and type of 2 is for
# change events.
eventid[0] = 23001;  type[23001]=2; #  New SSH User
eventid[1] = 23002;  type[23002]=2; #  New MAC for Host
eventid[2] = 22009;  type[22009]=2; #  User change detected
eventid[3] = 22010;  type[22010]=2; #  Server change detected
eventid[4] = 22011;  type[22011]=2; #  Device change detected
eventid[5] = 22012;  type[22012]=2; #  Network change detected
eventid[6] = 22021;  type[22021]=1; #  Compromise File Transfer
eventid[7] = 22020;  type[22020]=1; #  Suspicious File Transfer
eventid[8] = 20000;  type[20000]=1; #  Compromise Activity
eventid[9] = 23006;  type[23006]=1; #  Invalid Logon attemp
eventid[10] = 23007;  type[23007]=1; #  Suspicious-Windows_Event
count=10;

obj = object();
obj.callback.add("myCallback");
for (i=0; i<=count; i++)
   {obj.filter.event.id.add(eventid[i]);}
load_ip_file(file:ip_file);

function myCallback(obj)
{
   local_var src,dst,message;
   message = null;

   if ( (ip_list[obj.event.dst()] == 1) ||
             (ip_list[obj.event.src()] == 1) )
        {
         tm = localtime();
         timestring = tm["mon"] + "/" + tm["mday"] + "/"
                     + tm["year"] + " " + tm["hour"] + ":"
                     + tm["min"] + ":" +  tm["sec"];

         if ( type[obj.event.id()] == 2)
                {message_type = "SCADA-Policy_Event";}
         if ( type[obj.event.id()] == 1)
                {message_type = "SCADA-Intrusion";}

         message =  message_type + " - " + tm +
                   " a SCADA device had the following original log: "
                   + obj.event.data + '\n';

         obj.log(src:obj.event.src(), dst:dst, protocol:IPPROTO_TCP,
           sport:obj.event.sport(),dport:obj.event.dport(),
msg:message);
        }
}
```

Appendix D – scada_abuse2.tasl

```
include ('functions.inc');
global_var ip_list;
ip_list = null;

ip_file = '/usr/thunder/admin/SCADA.ip';

# Subscribed events. Type of 1 is for IDS events and type of 2 is for
# change events and 3 is for SCADA failover or shutdown events.
eventid[0] = 23001;  type[23001]=2; #  New SSH User
eventid[1] = 23002;  type[23002]=2; #  New MAC for Host
eventid[2] = 22009;  type[22009]=2; #  User change detected
eventid[3] = 22010;  type[22010]=2; #  Server change detected
eventid[4] = 22011;  type[22011]=2; #  Device change detected
eventid[5] = 22012;  type[22012]=2; #  Network change detected
eventid[6] = 22021;  type[22021]=1; #  Compromise File Transfer
eventid[7] = 22020;  type[22020]=1; #  Suspicious File Transfer
eventid[8] = 20000;  type[20000]=1; #  Compromise Activity
eventid[9] = 23006;  type[23006]=1; #  Invalid Logon attemp
eventid[10] = 23007; type[23007]=1; #  Suspicious-Windows_Event
eventid[11] = 450;  type[450]=3; #  OASyS-STANDBY_to_FAIL
eventid[12] = 451;  type[451]=3; #  OASyS-Automatic_SHUTDOWN
eventid[13] = 452;  type[452]=3; #  OASyS-Failover_Suceeded
count=13;

obj = object();
obj.callback.add("myCallback");
for (i=0; i<=count; i++)
   {obj.filter.event.id.add(eventid[i]);}

load_ip_file(file:ip_file);

function myCallback(obj)
{
   local_var src,dst,message;

   message = null;

   if ( (ip_list[obj.event.dst()] == 1) ||
             (ip_list[obj.event.src()] == 1) )
       {
        tm = localtime();
        timestring = tm["mon"] + "/" + tm["mday"] + "/"
                     + tm["year"] + " " + tm["hour"] + ":"
                     + tm["min"] + ":" +  tm["sec"];

        # if we are not SCADA, just do a log
        if (type[obj.event.id()] != 3)
        {

        if ( type[obj.event.id()] == 2)
               {message_type = "SCADA-Policy_Event";}
        if ( type[obj.event.id()] == 1)
               {message_type = "SCADA-Intrusion";}
```

```
        message =  message_type + " - " + tm +
                  " a SCADA device had the following original log: "
                  + obj.event.data + '\n';

        obj.log(src:obj.event.src(), dst:dst, protocol:IPPROTO_TCP,
          sport:obj.event.sport(),dport:obj.event.dport(),
msg:message);
        }

        # if we are a failover event, count the number of changes and
        # intrusion events in the last 15 minutes

        else
        {
        num_changes = 0;
        num_intrusions = 0;
        last15 = 900; # 900 seconds is 15 minutes
        last15ago = unixtime() - last15;

                for (i=0; i<=count; i++)
                        {
                        id = eventid[i];
                        theType = type[id];
                        amount = obj.event.count.get(id:id,

newer_than:last15ago);
                        if (theType == 1) {num_intrusions =+ amount;}
                        if (theType == 2) {num_changes =+ amount;}
                        }

        if ( (num_intrusions != 0) || (num_changes != 0) )
                {
                message = "SCADA-Suspicious_Failover - " + tm +
                          " a SCADA device went into failover mode‟ +
                          „ and this was proceeded by "
                          + num_intrusions + " correlated intrusion „
                          + „attempts and " +
                          + num_changes + " device changes";

                obj.log(src:obj.event.src(), dst:dst,
                        protocol:IPPROTO_TCP,
                        sport:obj.event.sport(),
                        dport:obj.event.dport(), msg:message);
                }
          }
    }
}
```

Appendix E – scada_abuse3.tasl

```
obj = object();
obj.filter.ip.src.add("0.0.0.0/0");
obj.callback.add("myCallback");

BRO_SENSITIVE              = 5306;
DRAGON_COMPROMISE          = 5006;
INTRUSHIELD_COMMAND_SHELL1 = 5276;
INTRUSHIELD_COMMAND_SHELL2 = 5277;
INTRUSHIELD_COMMAND_SHELL3 = 5278;
INTRUSHIELD_CODE_EXEC1     = 5245;
INTRUSHIELD_CODE_EXEC2     = 5246;
INTRUSHIELD_CODE_EXEC3     = 5247;
NEVO_TCP_COMPROMISE        = 5700;
NEVO_UDP_COMPROMISE        = 5701;
SNORT_USER_PRIV1           = 5113;
SNORT_USER_PRIV2           = 5136;
SNORT_ADMIN_PRIV1          = 5116;
SNORT_ADMIN_PRIV2          = 5139;
TP_MAJOR_ALERT             = 5758;
TP_CRITICAL_ALERT          = 5762;
TP_SMS_CRITICAL_ALERT      = 5802;
TP_SMS_MAJOR_ALERT         = 5806;
NETSCREEN_IDP_CRITICAL     = 6850;
NETSCREEN_IDP_MAJOR        = 6853;
ISS_OVERFLOW1              = 2601;
ISS_OVERFLOW2              = 2611;
CISCO_OVERFLOW1            = 2703;
CISCO_OVERFLOW2            = 2710;
CISCO_COMMAND1             = 2704;
CISCO_COMMAND2             = 2711;
CISCO_COMMAND3             = 2717;
FAILOVER1                  = 450;
FAILOVER2                  = 451;
FAILOVER3                  = 452;

obj.filter.event.id.add(BRO_SENSITIVE);
obj.filter.event.id.add(DRAGON_COMPROMISE);
obj.filter.event.id.add(INTRUSHIELD_COMMAND_SHELL1);
obj.filter.event.id.add(INTRUSHIELD_COMMAND_SHELL2);
obj.filter.event.id.add(INTRUSHIELD_COMMAND_SHELL3);
obj.filter.event.id.add(INTRUSHIELD_CODE_EXEC1);
obj.filter.event.id.add(INTRUSHIELD_CODE_EXEC2);
obj.filter.event.id.add(INTRUSHIELD_CODE_EXEC3);
obj.filter.event.id.add(NEVO_TCP_COMPROMISE);
obj.filter.event.id.add(NEVO_UDP_COMPROMISE);
obj.filter.event.id.add(SNORT_USER_PRIV1);
obj.filter.event.id.add(SNORT_USER_PRIV2);
obj.filter.event.id.add(SNORT_ADMIN_PRIV1);
obj.filter.event.id.add(SNORT_ADMIN_PRIV2);
obj.filter.event.id.add(TP_MAJOR_ALERT);
obj.filter.event.id.add(TP_CRITICAL_ALERT);
obj.filter.event.id.add(TP_SMS_CRITICAL_ALERT);
```

```
obj.filter.event.id.add(TP_SMS_MAJOR_ALERT);
obj.filter.event.id.add(NETSCREEN_IDP_CRITICAL);
obj.filter.event.id.add(NETSCREEN_IDP_MAJOR);
obj.filter.event.id.add(ISS_OVERFLOW1);
obj.filter.event.id.add(ISS_OVERFLOW2);
obj.filter.event.id.add(CISCO_OVERFLOW1);
obj.filter.event.id.add(CISCO_OVERFLOW2);
obj.filter.event.id.add(CISCO_COMMAND1);
obj.filter.event.id.add(CISCO_COMMAND2);
obj.filter.event.id.add(CISCO_COMMAND3);
obj.filter.event.id.add(FAILOVER1);
obj.filter.event.id.add(FAILOVER2);
obj.filter.event.id.add(FAILOVER3);

function myCallback(obj)
{

if ( (obj.event.id() >= 450) && ((obj.event.id() >= 452) )

{
buffer = unixtime() - 900;   # 15 minutes
count =  obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:BRO_SENSITIVE);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:DRAGON_COMPROMISE);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:INTRUSHIELD_COMMAND_SHELL1);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:INTRUSHIELD_COMMAND_SHELL2);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:INTRUSHIELD_COMMAND_SHELL3);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:INTRUSHIELD_CODE_EXEC1);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:INTRUSHIELD_CODE_EXEC2);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:INTRUSHIELD_CODE_EXEC3);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:NEVO_TCP_COMPROMISE);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:NEVO_UDP_COMPROMISE);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:SNORT_USER_PRIV1);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:SNORT_USER_PRIV2);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:SNORT_ADMIN_PRIV1);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:SNORT_ADMIN_PRIV2);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:TP_MAJOR_ALERT);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:TP_CRITICAL_ALERT);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:TP_SMS_CRITICAL_ALERT);
```

```
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:TP_SMS_MAJOR_ALERT);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:NETSCREEN_IDP_CRITICAL);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:NETSCREEN_IDP_MAJOR);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:ISS_OVERFLOW1);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:ISS_OVERFLOW2);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:CISCO_OVERFLOW1);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:CISCO_OVERFLOW2);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:CISCO_COMMAND1);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:CISCO_COMMAND2);
count += obj.event.count.get(dst:obj.event.src(), newer_than:buffer,
id:CISCO_COMMAND3);

if ( count > 0 )
    {
    message = "SCADA-Suspicious_Failover - host: " + obj.event.src()
              + " has attacked host: " + obj.event.dst() +
              " and it underwent a failover";

    obj.log(src:obj.event.src(), dst:obj.event.dst(),
protocol:IPPROTO_TCP,
            sport:obj.event.sport(), dport:obj.event.dport(),
msg:message);

    # filter him out for an hour
    obj.filter.ip.src.remove.until(ip:obj.event.src(),
                                   until:unixtime() + 3600);
    }
}
```

Appendix F – DNP3 Flood Detection Script

```
obj = object();
obj.filter.ip.src.add("0.0.0.0/0");
obj.callback.add("myCallback");
obj.filter.event.dport.add(20000);   # DNP3

function myCallback(obj)
{
local_var count, last5;
last5 = unixtime() - 300;
count = obj.event.count.get(src:obj.event.src(),
                            newer_than:last5);
if ( count >= 5000 )
   {
   message = 'SCADA-DNP3_FLOOD - ' + count +
              ' DNP3 events in last 10 seconds from ' +
obj.event.src();

   obj.log(src:obj.event.src(), dst:obj.event.dst(),
           sport:obj.event.sport(), dport:obj.event.dport(),
           msg:message );

   # Don't ever report this host again
   obj.filter.ip.src.remove.until(ip:obj.event.src(),
                                  until:unixtime () + 24*3600);
```