Example Custom LCE Log Parsing – Minecraft Server Logs

August 3, 2012

(Revision 1)

Ron Gula
Tenable CEO

Table of Contents

OVERVIEW

The LCE parses logs and normalizes events from many different types of servers, and more types are frequently added. However, every organization has logs from proprietary sources or devices that don't squarely fall into Tenable's focus of security devices, operating systems (OS), applications, and authorization sources. In this document, we'll look at the Minecraft application and discuss strategies for configuring LCE to normalize the data.

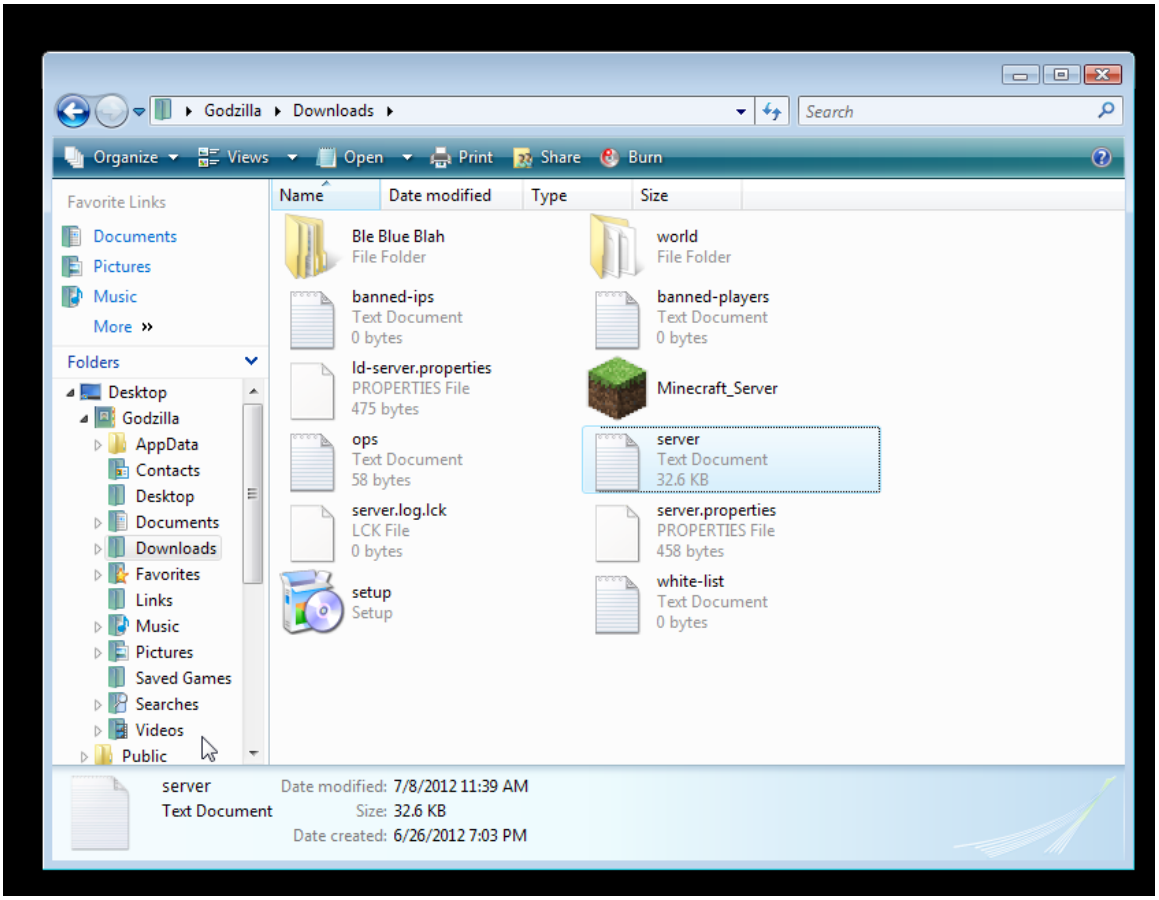
USING LCE TO NORMALIZE MINECRAFT DATA

Even though this example only adds four PRMs to parse Minecraft logs, the discussion about how and why logs were chosen, how the parsers were written, and what a user could do with these alerts is invaluable to any LCE administrator.

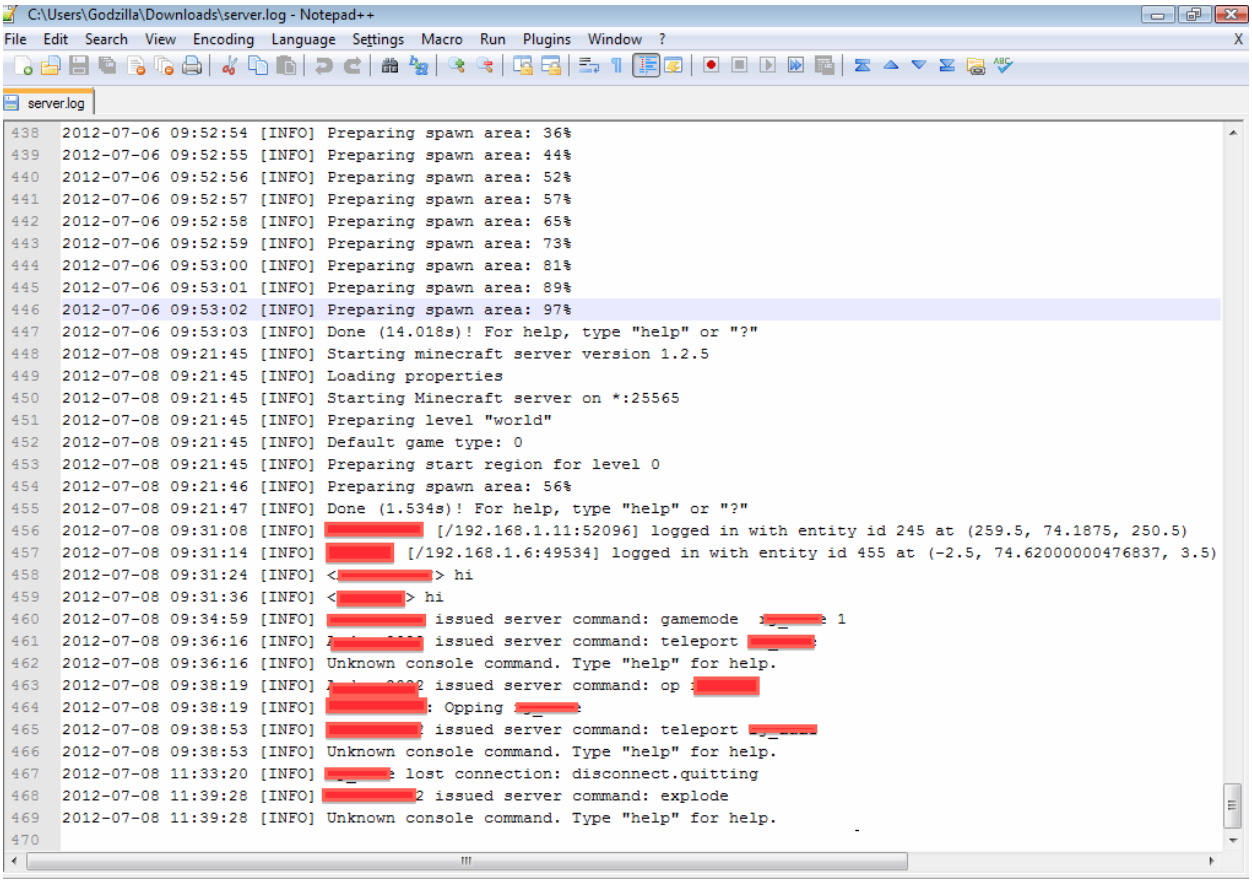
SETTING UP MINECRAFT LOGS

Minecraft is a popular Java-based application that allows users to create worlds and interact with other users. There are many public Minecraft servers to access, and it is really easy to set up a Minecraft server for use with your friends, family, or the general public.

I installed Minecraft on a Windows Vista system and the executable ran directly out of the installation directory as shown below:



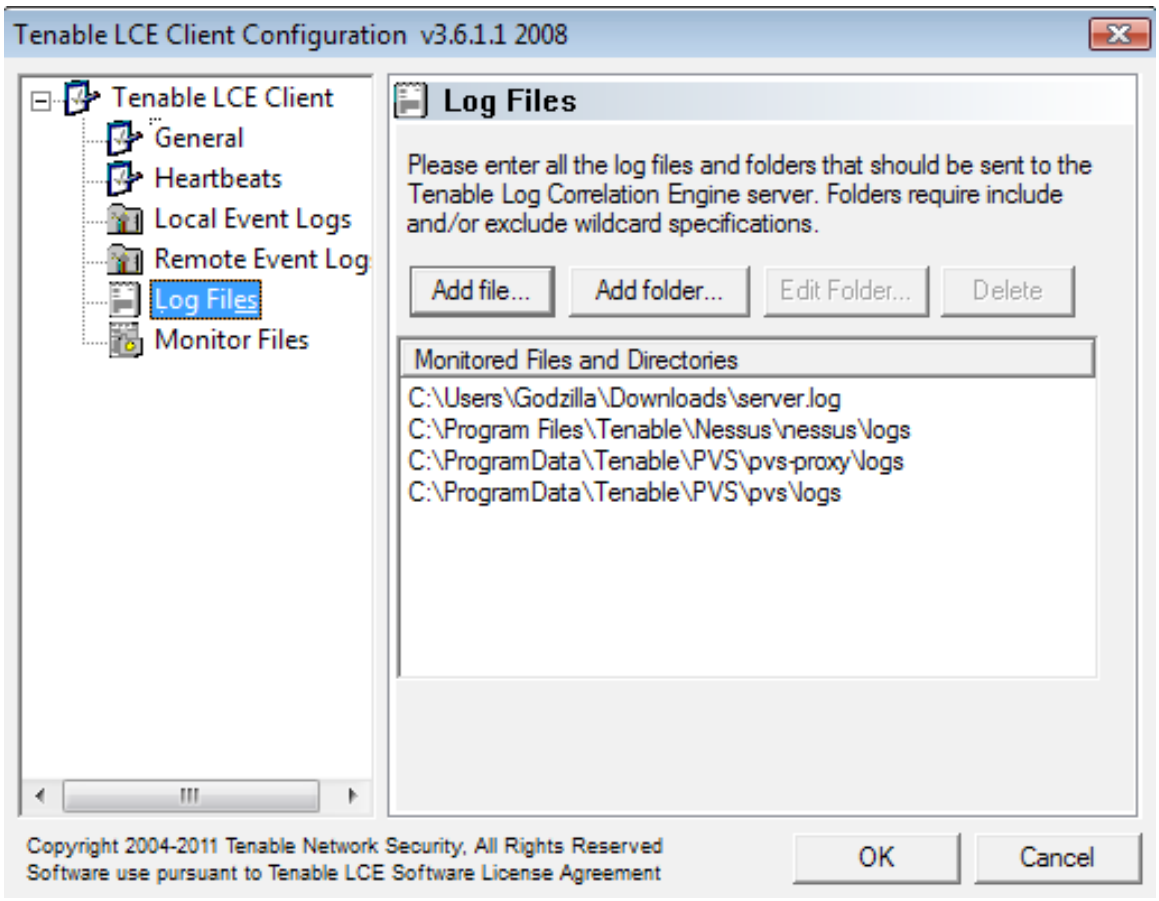
Once up and running, a single log file is created named **`server.log`**. This log had lots of interesting events in it and a sample screen shot is shown below:



Everything is sent to this file. Logs occurred for many reasons, including copies of an IRC style chat function, the ability for an administrator to "op" players, status events from the Minecraft server, login and logout events, and so on.

Configuring LCE to Monitor Minecraft Log

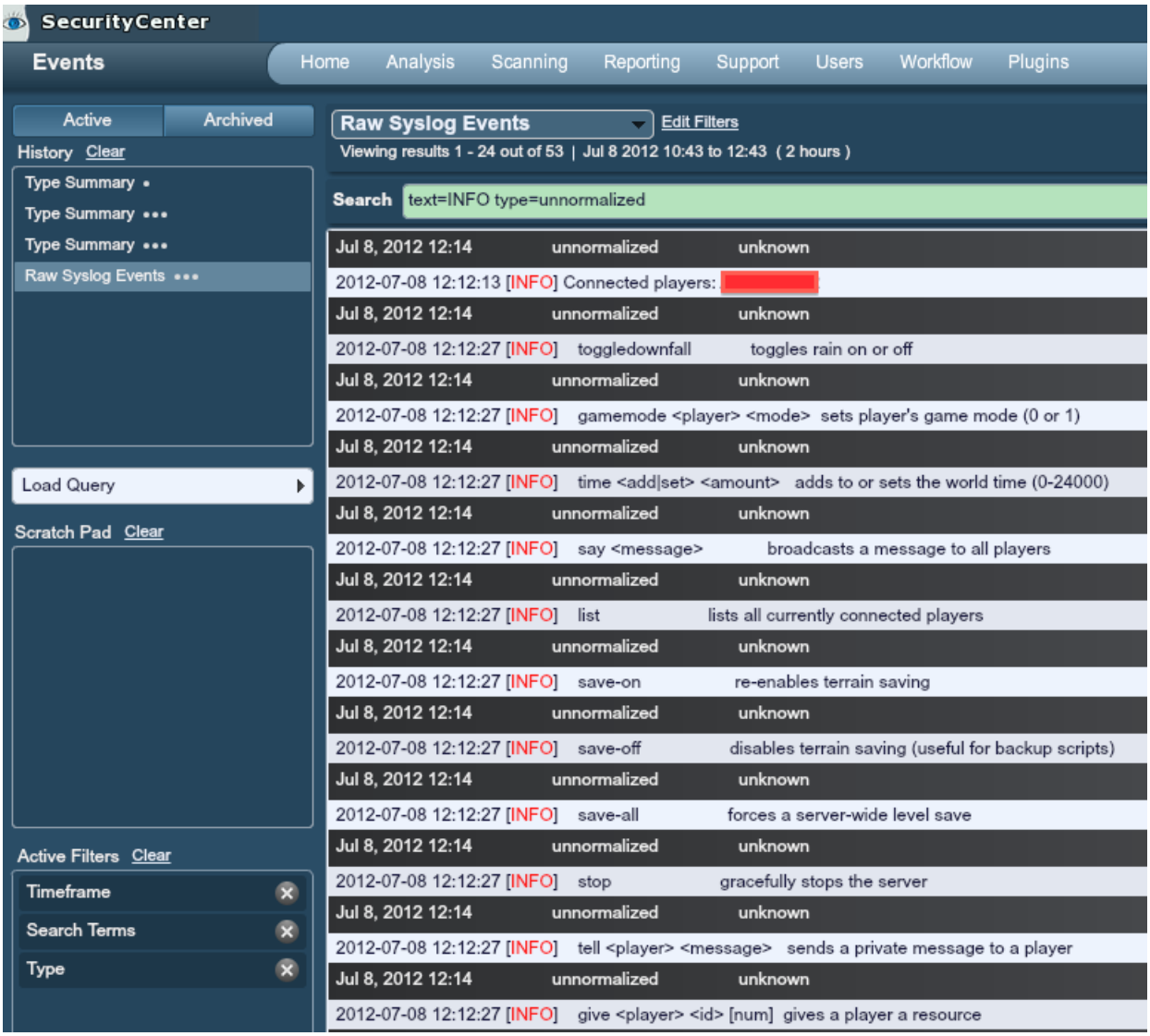
I configured a LCE Windows client to "tail" this file, as shown below:



I was also running Windows copies of Nessus and the Passive Vulnerability Scanner (PVS) on this system. The Minecraft log source is the first line, which references the **`server.log`** file.

Viewing Minecraft Logs with LCE

Once the LCE client was configured to tail this log, it started sending Minecraft logs to the LCE and they all went to the "unnormalized" event type, but were available for search right away as shown below:



In this case, the Minecraft logs don't have a lot of tags in them that could be used to anchor LCE pattern matching rules. This is one of the reasons I chose this as a log source. However, for my particular lab setting, no other unnormalized logs had the keyword "INFO" in them, so they were found with a query as shown.

WRITING A CUSTOM LOG PARSER

When Tenable Research engineers write parsers for applications, we try to identify many types of generic activity. Our goal isn't to create an application parser that identifies unique events for every possible event, but to create events that normalize other events so that every unique event can be quickly understood.

If we were officially supporting Minecraft logs, I would suggest to our Research that they focus on the following types of logs:

- **General system status events**
- **System errors**

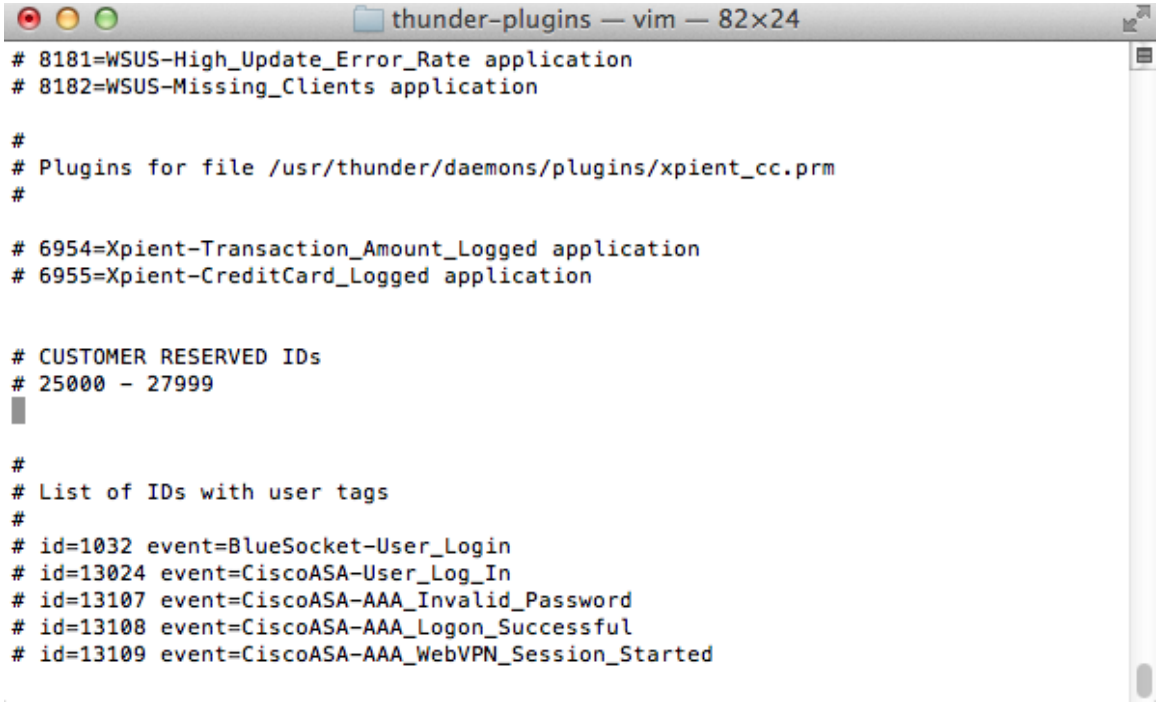
- **Authentication events**
- **Significant application events**

Most of the logs in the above screen shot don't fall into this category but would still be searchable with the LCE's full text search functions.

Parser Structure

So let's start writing a log parser. If you've read the LCE PRM documentation, you know that each LCE log parsing rule is required to have an ID, a Name, at least one pattern match statement, and a log statement that specifies the *name* of the event and the *type* of the event.

Which ID should we start with? In the **`prm_map.prm`** file located in **`/opt/lce/daemons/plugins`**, Tenable calls out the range 25000 through 27999 as IDs that are kept open for customers' custom PRMs. Below is a screen shot of this file:



This file also holds a list all other event names and types, so it's a great reference for choosing names and types that fit the LCE style.

One other note to discuss is which types to use. I've spoken with many customers who simply put all of the events for a certain application into a type category named after that event. In this case, it would be tempting to name the type "minecraft" and simply move on from there. However, Tenable suggests that types be associated with very high-level event names such as "login", "error", or "application" and that the names contain the actual technology.

The advantage to this approach is that as a SIEM, the LCE wants to provide context for a given event. Since we will be naming all of our Minecraft events with the name "Minecraft" in them, if a user wanted to see all of the Minecraft events, they'd simply type "Minecraft*"

into the text search screen. However, if they were analyzing login failures, system performance, network errors, network attacks, etc., they would be able work with higher level event types and see the Minecraft events in context alongside OS logs, network logs, etc.

So having said that, we'll start with event ID 25000 and use event types commensurate with the type of log we are parsing.

Logging Minecraft Server Starts

Scanning the example logs shows several entries that indicate that the Minecraft server had started. None of them are as obvious as the following though:

```
2012-07-06 09:53:02 [INFO] Starting minecraft server version 1.2.5
```

An extremely simple PRM for this log could be as follows:

```
id=25000
name=Minecraft Start
match=[INFO] Starting minecraft server version 1.2.5
log= name:Minecraft-Start type:restart
```

You may also be wondering about the "restart" category and where you would use that. The list of PRM types is included in most of the advanced documentation for LCE, including the LCE Best Practices and LCE Correlation papers. If you are command line type of person, you can also **`grep`** for "type" in the log file.

A log like this would indeed parse the example Minecraft log, but there are several issues with this. First, the "name" keyword is in fact the description that SecurityCenter or the Log Correlation Engine management console uses to display what the log means. Second, the match statement is too specific. If the version of Minecraft changes from 1.2.5 to something else, the match statement won't work. Let's modify things more as follows:

```
id=25000
name=A Minecraft server has started.
match=[INFO] Starting minecraft server version
log= name:Minecraft-Start type:restart
```

That's better. However, if you compare this to other Tenable PRMs for other logs, this looks really, really short. Tenable PRMs tend to have many, many more "match=" statements. There are two reasons for this – speed and flexibility.

The LCE builds pattern matching trees based on the frequency in which the pattern occurs in the entire body of all PRMs. As a unique pattern, our single match string is indeed unique to just our log. This causes the LCE to perform a string search of "[INFO] Starting minecraft server version" on every log. What we want to do is help the LCE to be smarter.

The string "INFO" is a good string to consider. If you **`grep`** the PRMs for "INFO", you'll see "match=IN" and "match=FO" in many examples such as plugin 8272 in the **`tenable_sc4_logs.prm`** file.

Rewriting to take advantage of this, our PRM would then look like:

```
id=25000
name=A Minecraft server has started.
match=[INFO] Starting minecraft server version
match=IN
match=FO
log= name:Minecraft-Start type:restart
```

It seems very counterintuitive, but since the patterns "IN" and "FO" are referenced so many other times, the LCE checks that pattern first and won't bother searching this plugin if it can't find either pattern.

How many patterns are enough? Tenable performs lots of statistical analysis on our repository of sampled and collected logs so they are very efficient. As long as your logs reference a handful of these shorter, more frequent matches, you should be fine.

From a pattern matching perspective, if you wanted to optimize the above log for speed, you should really be asking yourself how often the strings "IN" or "FO" show up in your regular logs. If you had millions of Check Point firewall logs that had a string in them such as "INBOUND firewall deny from 192.168.1.12" then the "IN" would match and the LCE would spend time searching for our long "match=" statement.

Using the **grep** tool and the **wc** (word count) utility can help you look for patterns that will accelerate your pattern matching performance on LCE. Below are some quick searches through the Tenable PRMs as of July 2012:

```
$ grep match=INFO *.prm | wc -l
      28
$ grep match=IN *.prm | wc -l
     243
$ grep match=FO *.prm | wc -l
      91
$ grep match=on *.prm | wc -l
     733
$ grep match=er *.prm | wc -l
     194
```

Adding these new patterns to our PRM would end up looking like this:

```
id=25000
name=A Minecraft server has started.
match=[INFO] Starting minecraft server version
match=IN
match=FO
match=er
match=on
match=INFO
log= name:Minecraft-Start type:restart
```

If you looked at the logs closely, there was a second Minecraft log that identified that the server was starting, and the port that the system was listening on. There would be nothing wrong with writing a second log for this if this was indeed important to you. However, if you are saving all logs, they are available for search even if you don't normalize them. The question is if you want two events that indicate Minecraft has started when perhaps one will do.

Logging a General Server Command

Consider the following log:

```
2012-07-08 09:34:59 [INFO] user567 issued server command: gameode  user223
```

In this case a user named "user567" seemed to have issued a command. We will do two things in this new PRM to watch for these events. First, we will match on the "INFO" and the "issued server command:". Second, we will extract "user567" as a user element.

```
id=25001
name=A Minecraft user has issued a server comment.
match=[INFO]
match=issued server command:
match=IN
match=FO
match=er
match=on
match=INFO
match=ss
match=se
match=ed
match=com
regex=INFO\] (.*) issued
log= name:Minecraft-Server_Command type:application user:$1
```

If you look closely, I added an additional "match=[INFO]" statement in anticipation of using this for every Minecraft PRM. That particular string isn't used at all in Tenable PRMs, but it's OK to have an anchor like that. We also added many other short two and three letter patterns that were contained in the "issued server command:" pattern.

Before our log statement, there is a regular expression that attempts to highlight the user name between the "[INFO]" string and the "issued" word. Since Minecraft usernames can't have spaces in them, were safe to use a generic ".*" to extract the user. If you aren't familiar with regular expressions, the parentheses around the ".*" are not part of the search pattern. Instead, they are used to assign a variable. In this case for our example log, they extract the string "user567" and assign it to the event.

Later on, once we have all of our Minecraft logs being sent to the LCE, any of the logs with the "user" keyword in them are used to associate a username with the event. This allows LCE users to summarize activity, list users, report on just one user, and other types of analysis.

Our log statement also chose a name of "Minecraft-Server_Command" and an event type of "application".

Logging Login Events

For our next example, we will consider a login event. Below is an example Minecraft login event:

```
2012-07-08 09:31:08 [INFO] user567 [/192.168.1.11:52096] logged in with
       entity id 245 at (259.5, 74.1875, 250.5)
```

There are lots of very good anchors in this log. Sometimes, logs are very sparse and there isn't anything to differentiate them with. There are hundreds of login PRMs in Tenable's set of available PRMs and we are very diligent about differentiating logins from devices so they can be automatically recognized by the LCE.

Having said that, consider the following PRM that focuses just on matching:

```
id=25002
name=A Minecraft user has logged into the server.
match=[INFO]
match=INFO
match=IN
match=FO
match=log
match=ed
match=in
match=lo
match=logged in with
match=entity id
log= name:Minecraft-User-Login type:login
```

These match statements have the appropriate high-speed matches, as well as anchors that distinctly map to "[INFO]", "logged in with", and "entity id".

The name and type for this log are also very straightforward.

However, with this type of log, we want to do two more things – extract the IP address of the login source and also extract our user ID. Any login or authentication log that has a user ID and a source IP can be used by the LCE to generically associate events to user IDs.

If we were working with Minecraft logs exclusively, this doesn't really help us because every one of their logs seems to have a user ID in it that we can extract. A modern network would have other logs sent to the LCE such as NetFlow, web browsing, DNS lookups, intrusion detection, etc. These logs don't have our Minecraft user ID in them. What we will do next though is extract the user ID and source IP of the login event and then briefly talk about how LCE can use this information to tag other logs automatically.

To extract the IP address and username, the regex statement is added to our PRM and the variables associated with it are also added. We'll also go a step further and extract the source port.

```
id=25002
name=A Minecraft user has logged into the server.
match=[INFO]
match=INFO
match=IN
match=FO
match=log
match=ed
match=in
match=lo
match=logged in with
match=entity id
regex=INFO\] (.*) \[\/ ([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+) \:([0-9]+) logged
log= name:Minecraft-User-Login type:login user:$1 srcip:$2 srcport:$3
```

The regular expression uses "\" characters to escape the occurrence of the various "[", "/" and ":" characters. The username, IP address, and source IP address of the log are all extracted to variables 1, 2, and 3, respectively.

If we wanted to configure LCE to leverage these logins as a trusted source of user information, we would configure plugin 25002 in the **`trusted_plugins.txt`** file located in **`/opt/lce/admin`** and then restart the LCE service.

Logging a General Minecraft Error

For our last example, we will identify an "error" in the example Minecraft logs and create a PRM for that. Tenable likes to identify errors in application logs for several reasons.

- **Tracking errors is useful in general system administration analysis**
- **Hackers often cause errors in systems when those systems are scanned or attacked**
- **The LCE's ability to highlight brand new errors, continuous errors, or statistical increases in errors is useful for general monitoring**

The example log we will use is:

```
2012-07-08 09:38:53 [INFO] Unknown console command. Type "help" for help
```

Before we parse this log, we should consider a basic problem of log parsing, which is not having access to the example logs in order to write a good parser. In the very small sample of logs we have for Minecraft, we've never seen any log that gave any sense of an error structure. Since we have lots of [INFO] tags, you may expect that we'd see something like [ERROR] or [ERR] in a log at some point. For this particular log, we'll assume that the "Unknown console command" is an error we want to track, but there are likely many other sources of errors. We could perform many different types of research to identify these logs. These include:

- Reading the documentation. The documentation often includes sample logs and discussions of log format. Tenable often tries to normalize production logs based on the vendor's nomenclature and log format.

- Searching the web for logs. If an admin has a bug in their system or a log they don't understand, they will often post these to discussion groups.
- Stress testing the application. One of the things Tenable does is to perform a Nessus scan on the application's ports, which can stimulate logs, login failures, protocol mismatch statements, etc.
- Keeping an eye on un-normalized logs. Once I have an application up and running, I typically run reports that leverage negative text search filtering to identify logs of interest that I hadn't seen before.

With that in mind, here is our PRM for the only real error we can see in the example logs:

```
id=25003
name=The Minecraft server encountered a user command that it didn't
       understand.
match=no
match=an
match=onsole
match=le
match=nknown
match=Un
match=mm
match=on
match=own
match=ommand
match=co
match=wn
match=now
match=ol
match=ma
match=command
match=INFO
match=[INFO]
match=IN
match=FO
match=Unknown console command
log= name:Minecraft-Unknown_Command type:error
```

There are many short patterns that are heavily used by other PRMs in the Tenable plugin set. I'm slightly concerned that the string "Unknown console command" sounds too generic. It's not a string that is in Tenable's vast database of vendor and application logs, but it does seem generic. If we wanted to tighten this up, we could modify it to match "[INFO] Unknown console command", which is more specific but is likely to match on its own. In very, very rare cases, Tenable has had to leverage a general regex rule that verified the date and time format as one of the only distinctions between similar logs from very different vendor sources.

Final PRM

Below is our "final" PRM.

```
# This is an example PRM to parse Minecraft logs that were written
       # against version 1.2.25
```

```
id=25000
name=A Minecraft server has started.
match=[INFO] Starting minecraft server version
match=IN
match=FO
match=er
match=on
match=INFO
match=[INFO]
log= name:Minecraft-Start type:restart

NEXT

id=25001
name=A Minecraft user has issued a server comment.
match=[INFO]
match=issued server command:
match=IN
match=FO
match=er
match=on
match=INFO
match=ss
match=se
match=ed
match=com
regex=INFO\] (.*) issued
log= name:Minecraft-Server_Command type:application user:$1

NEXT

id=25002
name=A Minecraft user has logged into the server.
match=[INFO]
match=INFO
match=IN
match=FO
match=log
match=ed
match=in
match=lo
match=logged in with
match=entity id
regex=INFO\] (.*) \[\/ ([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+) \:([0-9]+) logged
log= name:Minecraft-User-Login type:login user:$1 srcip:$2 srcport:$3


NEXT

id=25003
name=The Minecraft server encountered a user command that it didn't
       understand.
match=no
match=an
match=onsole
```

```
match=le
match=nknown
match=Un
match=mm
match=on
match=own
match=ommand
match=co
match=wn
match=now
match=ol
match=ma
match=command
match=INFO
match=[INFO]
match=IN
match=FO
match=Unknown console command
log= name:Minecraft-Unknown_Command type:error
```

CONCLUSION

Please post any questions about writing PRMs and general LCE "internal" operations to the Tenable Discussions Forum in the Log Correlation Engine area.

FOR MORE INFORMATION

Tenable has produced a variety of additional documents detailing the LCE's deployment, configuration, user operation, and overall testing. These documents are listed here:

- Log Correlation Engine Architecture Guide – provides a high-level view of LCE architecture and supported platforms/environments.
- Log Correlation Administrator and User Guide – describes installation, configuration, and operation of the LCE.
- Log Correlation Engine Client Guide – how to configure, operate, and manage the various Unix, Windows, NetFlow, OPSEC, and other clients.
- LCE High Performance Configuration Guide – details various configuration methods, architecture examples, and hardware specifications for achieving high performance with Tenable's Log Correlation Engine, specifically for organizations with logging requirements in the tens of thousands of Events Per Second (EPS).
- LCE Best Practices – Learn how to best leverage the Log Correlation Engine in your enterprise.
- Tenable Event Correlation – outlines various methods of event correlation provided by Tenable products and describes the type of information leveraged by the correlation, and how this can be used to monitor security and compliance on enterprise networks.
- Tenable Products Plugin Families – provides a description and summary of the plugin families for Nessus, Log Correlation Engine, and the Passive Vulnerability Scanner.
- Log Correlation Engine Log Normalization Guide – explanation of the LCE's log parsing syntax with extensive examples of log parsing and manipulating the LCE's **`.prm`** libraries.
- TASL Reference Guide – explanation of the Tenable Application Scripting Language with extensive examples of a variety of correlation rules.

- Log Correlation Engine Statistics Daemon Guide – configuration, operation, and theory of the LCE's statistic daemon used to discover behavioral anomalies.
- Log Correlation Engine Large Disk Array Install Guide – configuration, operation, and theory for using the LCE in large disk array environments.
- Example Custom LCE Log Parsing - Minecraft Server Logs – describes how to create a custom log parser using Minecraft as an example.

Documentation is also available for Nessus, the Passive Vulnerability Scanner, and SecurityCenter through the Tenable Support Portal located at https://support.tenable.com/support-center/.

There are also some relevant postings at Tenable's blog located at http://blog.tenable.com/ and at the Tenable Discussion Forums located at https://discussions.nessus.org/community/lce.

For further information, please contact Tenable at support@tenable.com, sales@tenable.com, or visit our web site at http://www.tenable.com/.

ABOUT TENABLE NETWORK SECURITY

Tenable Network Security, the leader in Unified Security Monitoring, is the source of the Nessus vulnerability scanner and the creator of enterprise-class, agentless solutions for the continuous monitoring of vulnerabilities, configuration weaknesses, data leakage, log management, and compromise detection to help ensure network security and FDCC, FISMA, SANS CAG, and PCI compliance. Tenable's award-winning products are utilized by many Global 2000 organizations and Government agencies to proactively minimize network risk. For more information, please visit http://www.tenable.com/.

Tenable Network Security, Inc.
7063 Columbia Gateway Drive
Suite 100
Columbia, MD 21046
410.872.0555
www.tenable.com