For-rest project

Jacques Gélinas
jacquesgelinas2407 at gmail.com

January 19, 2018

Abstract

The **for-rest** project is a **host based intrusion detection**. It can trigger an alarm whenever your server is doing something different from usual. Unlike other intrusion detection systems, it analyses the processes/programs hierarchy to identify intrusion patterns.

Introduction

Servers are robots. Unlike workstations, a server executes a limited set of daemons and processes repeatedly, forever. If we could record all the processes executed on a server, with enough information, one would be able to make a list of all usual process patterns. A pattern is like the **pstree** command. It represents the relation between a process, its parent, the parent of the parent and so on. Whenever a new pattern arises, it means something has changed in the server ... or it has been hacked.

Well, if we could record all the processes executed on a server, ... well, it turns out we can :-) Thanks to the **bsd process accounting**, much of the work is done.

Name of the project: for-rest

The name come from two ideas: The project analyses a big bunch of process patterns, organized in trees. Since there are so many trees, it is a forest. But the goal of this project is to have some confidence that our servers are not abused. As such, the goal is to sleep better, so the name is for-rest or for-a-better-rest.

Vservers

The vserver project has delivered since 2001 a container based virtualization/isolation solution to run multiple servers on the same host. One great value of this solution, compared with **real** virtual machines, is the ability for the host to interact (read spy) with the vservers for all kind of purposes. This includes maintenance, backups and monitoring. The vservers does not have to participate (to open services, enable sshd, etc...) to benefit from that. Actually, the vserver has no way to prevent it.

Principles

bsd process accounting

The bsd process accounting is an old solution available in all UNIX systems. It records at process end time various statistics. These were normally used to send an invoice to the users (time sharing system). Now this functionality is an oldy. Computing is cheap.

For each process, a log entry is created when the process ends. It holds at least the following information:

name This is the name of the program (not the full path).

flags Identify the type of records (see the **More log entries** section).

pid Process ID.

ppid Parent process ID.

uid User ID who executed that process.

gid Group ID of the user who executed this process.

btime Process startup time.

exitcode Execution results.

The kernel writes binary records to a file. Since it was meant for accounting, the kernel was only writing one record per process, only at end time.

Vservers and pid name-space

Vservers may be executed with their own PID name-space. It means that PID N on a given vserver is unrelated to PID N on another vserver. All vservers have independent process IDs, starting from 1 (init).

Each PID name-space may have its own process accounting. Process accounting is simply enabled by the **accton** utility. So enabling accounting at PID name-space creation will start to record all process activities in that name-space.

Vservers and the CAP_SYS_ACCT capability

Normally, vservers do not have the CAP_SYS_ACCT capability. As such, they can't control the process accounting. So a vserver ends up having process accounting enabled and can't do anything about it.

The accounting file

The accounting file is created outside of the vserver file system tree. So the vserver has no way to tamper with this file. It can't prevent the process logging from happening.

More log entries

When using process accounting as an intrusion detection input, we need information, not only at process end time, but minimally at process creation time, so we know when things are happening. We changed the kernel to write an accounting record here:

- At process creation time (when a fork() is executed). 0x20 is added to the flag.
- Every time a process exec() (and thus changes name). 0x40 is added to the flag.
- At process end time (we kept the original functionality).

With these new records, we can track process activity as they happen.

Process trees

The **forrest** utility reads the output of the kernel. The output file is specified by the **accton** utility. For each process ID, it finds the corresponding parent process. Then it performs the same for the parent process, finding the parent of the parent. It does that until it reaches the process one (init). This produces an execution tree. It uses slashes (/) to split the components. Each component looks like this:

```
program-name:verb other-name,uid,gid,exit-code
```

The **verb** is Fork, eXec or end. Note the use of upper case to simplify ordering. Fork, exec, end is the way process are started, and ended.

The exit-code is a 16 bit hex number. The first 2 hex digits represent the program exit code. The last 2 hex digits represent the signal number (if the program ended with a signal, 00 otherwise).

The other-name is there to help connect trees. Here is an example. We kept only the end of the lines to keep them short. Normally, all trees starts with the init process, followed by the startup scripts and so on.

```
.../ipop3d:Fork xinetd,0,0,0000
.../ipop3d:eXec xinetd,0,0,0000
.../ipop3d:end,0,12,000d
```

To make the trees a little easier to understand, we reversed the names. On the first line, we see that the process name **xinetd** has forked. This is how a new process is created on UNIX systems. Now, at this point, the process is still called xinetd. But to help connects trees together, we go a little in the future to find the corresponding eXec line. This line tell us the new name of the process. So we edit the Fork line to show the name **ipop3d**. In the eXec line, we go back in time to find the original name of the process (xinetd) and put it there. By using this name shifting, we end up with trees related to processes close together.

Here is a typical tree line, folded for readability:

```
init/init:Fork ,0,0,0000/rc:eXec capchroot,0,0,0000/S56xinetd:eXec rc,0,0,0000/
initlog:eXec S56xinetd,0,0,0000/xinetd:eXec initlog,0,0,0000/
xinetd:Fork ,0,0,0000/ipop3d:end,0,12,000d
```

Operation

Operations are done using three utilities:

forrest is the main utility to process and compare accounting files.

forrest-init creates a reference for a given vserver.

forrest-monitor is called every 5 minutes by cron and will check every running vserver.

Enabling process accounting

The process accounting is started using the **accton** utility. It has to be used early at PID name-space creation and before the CAP_SYS_ACCT capability is dropped. The **chcontext** utility has the –**pscript** option. It executes a script with full capabilities, just after the PID name-space has been created, but before the vserver is locked into its limited file system view. Hooked to the –pscript option, a script will create a new file in the **/var/run/vacct** directory. The file will be called **acct-vserver-name.log**.

The accounting file will simply grow, collecting process events.

The forrest utility

The **forrest** utility parses the accounting file and build a list of unique tree lines. It has three command line options:

–**facct** specifies an accounting file. The file will be read in memory. This option is always used.

–**build** specifies an output file name. It will contains a list of unique tree lines. The file produced by the build option is called the reference.

–**compare** specifies a reference file to read and compare with the 'in memory' trees produced by the –facct option. Any tree not found in the reference is printed on standard output. This indicates a new pattern has been detected. Time to investigate.

Creating a reference

To create a reference, you just starts a vserver and let it run for a while. You assume that the vserver has not been hacked. You simply issue the following command:

```
forrest-init vserver-name
```

This will create a file called **/var/lib/forrest/vserver-name.ref**. You can review this file. It contains all unique tree lines. Proof reading that file will confirm that the server is indeed doing what it is supposed to do (and no more).

Once the reference is created, this becomes the baseline. As the functionality of the vserver evolves, you will probably add lines to the reference, or simply re-create it from scratch using forrest-init.

Monitoring

Just put the forrest-monitor in the root cron, called every 5 minutes. It will loop through all running vservers. For each one, it will review the accounting file and compare it to the reference. Every tree line not found in the reference will trigger an alarm.

All offending lines will be place in the file **/var/log/forrest/vserver-name.tmp**. Normally those files are empty. Whenever there is an issue, you can review the files. If you agree that the new tree lines are indeed normal, you can append the file to the reference.

The /etc/forrest-monitor-emails.lst file

This file is optional. It contains a list of email addresses. If an issue is found by the monitoring tool, an email will be sent to all addresses found in that file, with the content of the /var/lib/forrest/forrest-vserver-name.tmp file.

The /var/lib/forrest/exclude-vserver-name.lst file

While vservers are robots (and always perform the same tasks), once in a while, they are maintained. The administrator will perform various tasks such as editing configuration files. This is very different from the processes usually exexcuted by the server (a server never executes **vi** on its own :-)).

This will trigger alarms. The file **/var/log/forrest/vserver-name.tmp** will contain the offending tree lines. It is possible to filter out the lines you know are ok using regex pattern. While you can't tell everything an administrator will do to maintain a vserver, there is a way to make this predictable.

You can write those patterns in the file exclude-vserver-name.lst. Enter one pattern per line. The pattern are regex. When forrest-monitor runs, any tree lines that match one of those pattern will be discarded. It won't be compared with the reference, so won't trigger an alarm.

Using vserver enter Most vserver do not need an sshd access. They are simply maintained from the host. An admin simply **enter** the running vserver and performs his work. All processes executed this way will start by **enter/enter** sub-string.

Using ssh If you are maintaining the vserver using ssh, then log as a normal user and then **su** your way to root. It will be easy to filter out everything executed under sshd by this user.

Restart your vserver You can also play safe. After every maintenance, you restart your vserver. You end up with a clean state and all tasks performed during the maintenance are gone.

Testing your exclude files

After having configured your exclude files with patterns, you end up with a clean state: No alarm anymore. Everything is under control ... or not. Maybe your regexs are too broad. Remember that your regexs are substrings. They may match surprising stuff. Even if they do not contain special regex characters, they are nevertheless matching any sub-string in a process pattern.

Once in a while, to make sure that you are not missing anything important, you can use the –**showexcluded** command line option to print (on stderr) all patterns which are matching one of the exclude regex. The lines printed

- do not appear in the reference file.
- won't contain any duplicates.

Here is a typical command line to find what is being filter out. The will /tmp/file-to-review.txt will contain excluded patterns.

```
forrest --showexcluded --facct /var/run/vacct/acct-VSERVER.log \
--compare /var/lib/forrest/VSERVER.ref \
--excludefile /var/lib/forrest/exclude-VSERVER.lst  2>/tmp/file-to-review.txt
```

Implementation

The kernel patch

You need a very simple patch on top of the kernel vserver patch. The patch is available here

```
https://solucorp.solutions/projects/forrrest/forrest-kernel-patch.diff
```

The forrest source code

The source code is available using **subversion** at

```
https://solucorp.solutions/repos/solucorp/forrest/trunk
```

You can build it simply by issuing

```
make
make install
```

or on fedora

```
make buildrpm
```

To compile it, you need the linuxconf-devel and linuxconf-lib package. You can grab the latest source here

```
https://solucorp.solutions/repos/solucorp/linuxconf/trunk
```

then you can build an rpm for it

```
make buildrpm
```

Linuxconf is not maintained anymore, but the library is. At some point it will be renamed...

Vserver tools

You also need the latest vserver package available through subversion at

```
https://solucorp.solutions/repos/solucorp/vserver/trunk
```

Hopefully the idea will be adapted in util-vserver as well.

Licence

This software is free and published under the GPL licence.