Global Information Assurance Certification Paper

Using CFENGINE to maintain systems and security.

Mike Lang
March 2001

CFENGINE (site configuration engine) was written by Mark Burgess at Oslo College, Norway. Its purpose was to free the system administrator in a Unix environment to do other tasks by automating tedious duties. In its simplest form it is a central configuration server for Unix hosts. It has two parts: (1) a configuration file that defines how a system should look and (2) a software agent that tries to bring a system in to convergence with the definition.

In the abstract it may be thought of as an immune system for a network of hosts. Based on the current system status cfengine can respond appropriately and act with out involving a system administrator. In maintaining Unix systems, most administrators go though stages. First an administrator configures hosts by hand. Next, as the number of machines increase, an administrator scripts the host configurations. Then when large numbers or heterogeneous hosts hit, he or she begins to rethink career choices. This tool addresses this problem by allowing you to spend your time on one good centrally maintained configuration file for all your hosts. Cfengine also has security in mind. As it runs, it watches for and logs any suid files and directories with names strange names like "…". These are things that a good system administrator would look for as they navigated the system.

If you have ever used an automated installation program such as jumpstart for Solaris or kickstart for Linux, you know how much time this can save. You build the configuration for your site once, then new installs or rebuilding damaged hosts takes minute of your time rather than hours. The automated installation servers will get a host to a known state, but will not keep it there. If you modify you jumpstart configuration you will have to change the installed base of hosts by hand or with a custom script. With cfengine the configuration can evolve with new information on exploits or security policy changes. There is no need to learn a different tool for each OS since cfengine can make decisions based on the OS type. With a good configuration file you can just take the default installation from the vendor and have cfengine configure the rest.

Installation:

To use this tool first you have to download it from http://www.iu.hio.no/cfengine/ or one of the mirrors. Cfengine is written in the c language to be light on system resources. It compiles easily with configure, make, and make install. (For details see the INSTALL file that comes with the distribution.) In version 1.6.3 you will need Berkley DB version 3.2 in order to use the MD5 checksum feature discussed later. There are different ways to run cfengine, the documentation suggests running it on the clients hourly. Also you will want cfd running on all the clients. Cfd allows you to request the client to run cfengine. This is useful if you need to defend against a new exploit now rather than waiting for the next cron job to hit. The cfd daemon also can be run on the server to allow file transfers. Keep in mind, the hourly run time needs when writing configuration files. You don't want your scripts to take more than an hour to run and can always specify special upkeep at low usage times; say 2AM Saturday.

Before I show some examples lets look at cfengine syntax. The cfengine configuration is a programming language [7].

- White space is ignored
- Comments start with a #.
- Words with a single colon are sections. (Only members of the predefined sections set are allowed).
- Words with double colon are classes. These are used to make decisions.
- *Word = (list)* is an assignment statement.

So a config file in it simplest form would be as follows:

```
control:
      actionsequence = ( links )
      links:
         /usr/local/bin/netscape -> /usr/local/bin/netscape.476
```

This example would create a symblolic link from the latest installed netscape version to to /usr/local/bin/netscape. The *actionsequence* defines what sections will be executed and in what order. In the example above only the *links* section will be executed. (Notice the spaces around the *links* in the *actionsequence*, these were required to get cfengine to run.) There are 21 predefined section names in version 1.6.3. They are designated with a single colon in the configuration file and are listed below [7]:

groups, control, homeservers, binservers, mailservers, mountables, import, broadcast, resolve, defaultroute, directories, miscmounts, files, ignore, tidy, required, links, disable, shellcomands, edifiles, and processes.

Classes are used to make decisions. There are hard classes and user defined classes. The Solaris class is referred to as a hard class, it is predefined by cfengine. Other predefined classes include the unqualified host names and time/date designations. To define your own classes, use the *groups* section.

```
groups:
      diskservers = ( ray, beam, spot, flare )
      testhosts = ( alpha, beta )
```

So you might delete old core files on your disk servers and test new cfengine configs on hosts alpha and beta. The classes can then be combined to make complex decisions. A "." Performs a logical AND between the classes, the "|" performs an OR and the "!" performs a NOT. So to kill the web server on you Linux test hosts every Friday:

```
processes:
      testhosts.linux.friday::
            "httpd"      signal=kill
```

When something can't be easily accomplished in cfengine the "shellcomands" section can be used to run custom Perl or shell scripts.

Now that we have the basics of what a configuration looks like, lets see how this tool can help us maintain the three pillars of security: (1) confidentiality, (2) integrity and (3) availability.

Confidentiality:

Confidentiality requires that we keep private information private. Lets investigate a configuration to guard one of unix system's most valuable items, passwords. The shadow password file is supposed to keep our encrypted passwords private so they can't be copied off to a remote host and cracked. The configuration file below will check the permissions and ownership of the shadow password file for any Linux hosts. If they are incorrect the *fixall* will cause cfengine to correct them.

```
control:
actionsequence = ( files )
   files:
      linux::
            /etc/shadow  mode=400 owner=root action=fixall
```

Keeping up with security holes is a big problem for system administrators. To illustrate closing a security hole lets consider a real example. One exploit that came up a while back was Solaris's sadmin daemon [4]. To close this, we want to check all Solaris hosts to make sure they are not configured as a service and to kill any existing daemons. So in cfengine this would look like the following:

```
control:
      actionsequence = ( disable, editfiles, processes )
      #cert advisory CA-1999-16
      disable:
         solaris::
            /usr/sbin/sadmind
      editfiles:
         solaris::
            {/etc/inetd.conf
               HashCommentLinesContaining  "sadmin"
            }
      #kill any running sadmind's and restart inetd
      processes:
            solaris::
               "sadmind"      signal=kill
               "inetd"        signal=hup
```

The three sections are disable, editfiles and processes. The action defined will be taken if the os matches the class "solaris". Any files listed in the *disable* section will be renamed to *file*.cf-disabled and the permissions will be set to 400[1]. Under *editfiles* the system will check inetd.conf to see if the sadmind service is configured, if there are any matches in inetd.conf, they will be commented out. Now if at a later time an old /etc/services file is restored, the system will correct itself at the next running of cfengine. When a new exploit is discovered the cfengine configuration is modified and all the hosts will update themselves.

Integrity:

To maintain your system integrity we must have a way to ensure the files and binaries are what we expect them to be. Cfengine can check for changes to system files. In this way it can also act as an intrusion detections system (IDS).

Tripwire [7] has been used for IDS and file integrity tool since 1992. Cfengine can be used to get a similar functionality. A database of MD5 checksums can be established files you want to protect, and then these can be compared to the checksums of the files actually on the system during subsequent runs of cfengine. Make sure you include executables that are usually replaced when crackers install rootkits on your system (telnetd, ftpd, logind, sshd, etc .). One of the philosophies in cfengine is to fix rather than just inform, so if a problem is found the binary should be replaced with a good copy from a read-only source or protected server. A CD of these important programs could be created from the OS distribution media and kept online for such updates.

Take sshd as an example. If a cracker breaks in a system and replaces sshd with a Trojan that collects accounts, passwords and machine names, this can be devastating to the entire enterprise. Here is how cfengine could be used to minimize the damage.

```
control:
actionsequence = ( copy )
   copy:
     solaris::
     # copy sshd from a known source
          /mnt/cdrom/sshd dest=/usr/local/bin/sshd
             type = checksum
             inform = true
             syslog = true
             mode=500
             owner=root
```

The option *type=checksum* causes the MD5 check to be done to see if the copy is needed. Permission and owner will also be set and inform and syslog will make sure the events are logged. We are assuming that a trusted binary has been written to a cdrom and is mounted on /mnt/cdrom, and that the MD5 database has been created previously. The old binary would be kept as sshd-cf.saved, this is useful for later forensics.

Other tools have been used to keep configuration files up to date. The two most common would be NIS, and rdist[5]. Using the copy section cfengine clients can request new files from the server to install new versions or replace damaged ones. NIS has been a troublesome security hole. If the NIS domain can be guessed, it can be fooled into giving your password file and other important configuration files away. In the example below we copy the group file to all hosts and set the proper permissions.

```
control:
actionsequence = ( copy )
   copy:
      any::
         /usr/local/cfengine/share/clientfiles/group.clients
                  dest=/etc/group
                  mode=444 owner=root
                  server=serverhostname
```

Availability:

To enhance availability we want to be proactive in monitoring resources and services. In this way we can monitor daemons and disk space, then act based on the discoveries. If you have a syslog server, you can also monitor your clients as they run hourly from cron. Set variable syslog to on for logging.

```
syslog = (on)
```

Here is an example to demonstrate how to keep your web services running.

```
groups:
  webservers = ( www.dom.com, www2.dom.com,
                 www-internal.dom.com )
control:
  actionsequence = ( processes )
processes:

 linux.webserver::
    "httpd" restart "/etc/rc.d/init.d/httpd start" #RH6.2
          useshell=false

 solaris.webserver::
    "httpd" restart "/etc/init.d/httpd start"
          useshell=false
```

First we will define a new class, webservers. Then using the *process* section we will restart the web daemons with the script in the init.d directory. Just to show how classes can be combined, we will take care of the differences between directory placement on RedHat 6.2 and Solaris. Notice the *useshell = false*. This is an added security feature you can use when running scripts. This starts programs directly without using an intermediate shell. Useshell guards against IFS attacks (when an evil doer tries to trick a program running as root into running an arbitrary command by changing the character used for command separation). Now that we have talked about keeping services running, the other side of the coin is to keep some service from running. Quite often you don't want client systems running ftpd, telnetd and other servers. You can also kill any known cracker tools and irc clients, such as eggdrop, IRC client BitchX etc. You would want to check for name collisions with your local programs to keep from killing valid programs.

```
processes:
      all::
            "autorun"    signal=kill
            "eggdrop"    signal=kill
            "irc  "      signal=kill
            "BitchX"     signal=kill
            "README"     signal=kill # you don't sh README!
      !webserver::
            "httpd"      signal=kill # no unauthorized webservers.
```

Notice above if the host is not in the defined group webservers, we want to kill any httpd processes on the host.

Other Possible Applications:

Dual Boot Machines:

Dual boot machine are problem for most system administrators. It is difficult to get changes to a machine when you don't know which operating system it is currently running. This is especially true for folks that boot the alternate system infrequently. What you end up with are hosts that are way behind on security fixes dangling off of you network. This is where automation systems that push out changes to hosts like Microsoft's SMS fail. With cfengine the host updates itself with a pull, so when an infrequent linux user boots his system, cfengine can be run to grab any changes that have been missed while the system was running an alternate operating system

Securing remote dial up machines for telecommuters

Dial up users are large security holes for any network. Cfengine would allow updating security settings for users at home. You could allow users to initial there own updates, or be more draconian and force it as part of the dial up or VPN procedure. To speed up the update process on these systems, rsync can be used [9]. Rsync allows just the changes in files to be transferred rather than the entire file. This is very useful due to the limited bandwidth on dialup connections.

Creating a patch servers

If you don't have direct control over all hosts on your network you can at least provide a patch repository to help secure your neighbors. In this way any host on the network can update to the most current tested patches with a simple run of cfengine. This same patch server could also be used as part of your own cfengine scripts.

Miscellaneous Configuration Notes:

A few security precautions; cfengine by default doesn't encrypt any of the traffic between hosts. If the files do need to be encrypted for transmission, this can be done with an option to the *copy* control.

```
copy:
    source dest=destination secure=true server=trusted
```

This would cause the file in question to be transferred via triple DES. The included program cfkey would be used to generate a key that is distributed to the clients. Other security options you may want to turn on include:

```
  CheckIdent = ( on )    # to check for spoofers
  SecureInput = ( on )   # make sure file is owned by cfengine
owner and not group/world writeable.
```

An interesting note, in the *process* section make sure your entries don't match the cfengine process, or its configuration file, I wrote one test that would kill itself before it got any work done.

Conclusions:

Cfengine addresses the biggest security hole faced today, the lack of time system administrator have to work on the systems. By automating repetitive tasks, and keeping watch on system files and resources, this tool can give computer staff more time to build a more resilient infrastructure. With this automation you can attack one of the basic security problems the herd effect [3]. Crackers only have to find one weak host on a network to infect the rest. With cfengine you can keep your herd at the same level. The documentation is extensive with a 100 page reference manual [9] and a 96 page tutorial[8], there are also example configurations that are included in the distribution. Though my examples are all very short, configurations files can get very large and should be kept up with a version control system such as RCS or CVS etc. Since cfengine is evolving there are always fixes, enhancements, and security bulletins, so joining the mailing lists and keeping up with changes is important. (http://www.iu.hio.no/cfengine/cfcomment.html).

References:

[1] Burgess, Mark. "Managing Network Security With Cfengine", 03 Jan, 2001.
URL: http://www.iu.hio.no/cfengine/cf-security.htm

[2] Burgess, Mark. "Computer Immune Systems", 01 Mar, 2001.
URL: http://www.iu.hio.no/~mark/research/immune.html

[3] Perrine,Tom. "Security as Infrastructure", 11 Dec, 1998. URL:
http://www.sdsc.edu/~tep/Presentations/1998.LISA.Security.Infrastructure/index.htm

[4] CERT/CC. "CERT® Advisory CA-1999-16 Buffer Overflow in Sun Solstice
AdminSuite Daemon sadmind ", 14 Dec, 1999.
URL: http://www.cert.org/advisories/CA-1999-16.html

[5] Ryan, Ron "Security and System Maintenance Automation**",** 22 Jan, 2001.
URL: http://www.sans.org/infosecFAQ/unix/automation.htm

[6] Tripwire Security Systems Inc, "Tripwire Academic

Source Release 1.3.1 User
Manual", 30 Apr, 1999.
URL: ftp://coast.cs.purdue.edu/pub/tools/unix/ids/tripwire/Tripwire-1.30-docs.pdf

[7] Burgess, Mark. "cfengine tutorial", 01 Dec, 2000.
URL: http://www.iu.hio.no/cfengine/docs/cfengine-Tutorial.html

[8] Burgess, Mark. "cfengine reference", 01 Dec, 2000.
URL: http://www.iu.hio.no/cfengine/docs/cfengine-Reference.html

[9] Davis, Jim. "Cfengine-and-rsync.htm", 06 Mar, 2001
URL: http://www.cs.arizona.edu/people/jdavis/cfengine.html