The AIDE manual

About this document

This manual is by no means complete, usable, readable, comprihensible, or error free.

If you have any corrections, additions or constructive comments, please report them as bugs, patches or feature requests here.

This document was originally written by Rami Lehti <rammer@cs.tut.fi> with additions made by Marc Haber <mh+aide-manual@zugschlus.de> and Richard van den Berg <richard@vdberg.org> .

Table of Contents

What is AIDE?

AIDE (Advanced intrusion detection environment) is an intrusion detection program. More specifically a file integrity checker.

AIDE constructs a database of the files specified in AIDE.conf, AIDE's configuration file. The AIDE database stores various file attributes including: permissions, inode number, user, group, file size, mtime and ctime, atime, growing size, number of links and link name. AIDE also creates a cryptographic checksum or hash of each file using one or a combination of the following message digest algorithms: sha1, sha256, sha512, md5, rmd160, tiger (gost and whirlpool can be compiled in if mhash support is available). Additionaly, the extended attributes acl, xattr and selinux can be used when expliticly enabled during compile time.

Typically, a system administrator will create an AIDE database on a new system before it is brought onto the network. This first AIDE database is a snapshot of the system in it's normal state and the yardstick by which all subsequent updates and changes will be measured. The database should contain information about key system binaries, libraries, header files, all files that are expected to remain the same over time. The database probably should not contain information about files which change frequently like log files, mail spools, proc filesystems, user's home directories, or temporary directories.

After a break-in, an administrator may begin by examinining the system using system tools like ls, ps, netstat, and who --- the very tools most likely to be trojaned. Imagine that ls has been doctored to not show any file named "sniffedpackets.log" and that ps and netstat have been rewritten to not show any information for a process named "sniffdaemond". Even an administrator who had previously printed out on paper the dates and sizes of these key system files can not be certain by comparison that they have not been modified in some way. File dates and sizes can be manipulated, some better root-kits make this trivial.

While it is possible to manipulate file dates and sizes, it is much more difficult to manipulate a single cryptographic checksum like md5, and exponentially more difficult to manipulate each of the entire array of checksums that AIDE supports. By rerunning AIDE after a break-in, a system administrator can quickly identify changes to key files and have a fairly high degree of confidence as to the accuracy of these findings.

Unfortunately, AIDE can not provide absolute sureness about change in files. Like any other system files, AIDE's binary and/or database can also be altered.

Compiling AIDE

I'm in a hurry. Bottomline about compilation.

After you have installed all the necessary sofware do `./configure;make;make install` in the main AIDE directory of the unpacked source tree. **You should carefully think about the configuration and what a possible hacker can do if he/her/they/it has root access.**

Getting all that is needed

Before you can compile AIDE you must have certain things.

- ANSI C-compiler (GCC will do just fine)
- GNU Flex
- GNU Bison
- GNU Make
- AIDE source code
- Mhash library
- And if you want to use postgres sql for database storage you must have the postgres sql developer library installed

Please check to see if there are mirrors available.

Once you have the source code of AIDE you should unpack it. If you have GNU tar then the command is `tar zxvf aide-version.tar.gz` .

Compile-time configuration

Next you must use the configure script found in AIDE's source code package to configure the compilation process.

There are several options you can give to configure. You can find out what options are available with `./configure --help` command. Most of the time you do not need to give any options. You can just use configure without any parameters.

If you want to use the bundled gnu regular expression package you can give the `--with-gnu-regexp` option. Some OS's that a buggy regexp implementation you must use this option.

If you want to change the directory where AIDE is installed you can use --prefix option. For example `./configure --prefix=/usr`

Compilation and installation

The compilation is done by simply typing `make`. You can now type `make install` to install the binary and the manual pages. The binary however should be installed on read-only media or in some other tamperproof place. Also the databases should be kept somewhere where a possible intruder cannot change them.

Configuration

Next you have to create a configuration file. You can find more documentation for this in aide.conf(5) manual page.

There are three types of lines in aide.conf:

- configuration lines - used to set configuration parameters and define/undefine variables
- selection lines - indicate which files will be added to the database
- macro lines - define or undefine variables within the the config file

Lines beginning with # are ignored as comments.

Here is an example configuration.

```
#AIDE conf

   # Here are all the things we can check - these are the default rules
```

```
#
#p:      permissions
#i:      inode
#n:      number of links
#l:      link name
#u:      user
#g:      group
#s:      size
#b:      block count
#m:      mtime
#a:      atime
#c:      ctime
#S:      check for growing size
#I:      ignore changed filename
#md5:    md5 checksum
#sha1:   sha1 checksum
#sha256: sha256 checksum
#sha512: sha512 checksum
#rmd160: rmd160 checksum
#tiger:  tiger checksum
#haval:  haval checksum
#crc32:  crc32 checksum
#R:      p+i+l+n+u+g+s+m+c+acl+selinux+xattrs+md5
#L:      p+i+l+n+u+g+acl+selinux+xattrs
#E:      Empty group
#>:      Growing logfile p+l+u+g+i+n+S+acl+selinux+xattrs
#The following are available if you have mhash support enabled:
#gost:   gost checksum
#whirlpool: whirlpool checksum
#The following are available when explicitly enabled using configure:
#acl:    access control list
#selinux SELinux security context
#xattr:  extended file attributes

# You can alse create custom rules - my home made rule definition goes like this
#
MyRule = p+i+n+u+g+s+b+m+c+md5+sha1

# Next decide what directories/files you want in the database

/etc p+i+u+g     #check only permissions, inode, user and group for etc
/bin MyRule      # apply the custom rule to the files in bin
/sbin MyRule     # apply the same custom rule to the files in sbin
/var MyRule
!/var/log/.*     # ignore the log dir it changes too often
!/var/spool/.*   # ignore spool dirs as they change too often
!/var/adm/utmp$  # ignore the file /var/adm/utmp
```

Here we include files in /etc, /bin and /sbin. We also include /var but ignore /var/log, /var/spool and a single file /var/adm/utmp.

It is generally a good idea to ignore directories that frequently change, unless you want to read long reports. It is good practice to exclude tmp directories, mail spools, log directories, proc filesystems, user's home directories, web content directories, anything that changes regularly. It is also good practice to include all system binaries, libraries, include files, system source files. It will also be a good idea to include directories you don't often look in like /dev /usr/man/.*usr/. Of course you'll want to include as many files as practical, but think about what you include.

One example: If you have a block device whose owner is changing frequently, you can configure aide to just check the attributes that do not normally change (inode, number of links, ctime).

Note that if you are referring to a single file you should add $ to the end of the regexp. This matches to the name of the file exactly and does not include any other files that might have the same beginning. In the example, all filenames beginning with /var/adm/utmp would be ignored if there were no dollar sign at the end of the last line. An intruder could then create a directory called /var/adm/utmp_root_kit and place all the files he/she/they wanted there and they would be ignored by AIDE.

There are two special group definitions to tweak what attributes are printed in the report. First report_attributes lists those attributes that are always printed from changed files. For example, if you say

```
attributes = u+g
```

and the size of a file changes, it's user and group id will also be printed in the report. Secondly, ignore_list defines which attributes to ignore from the report. For example, if you define

```
ignore_list = b
```

and this size of a file changes, it's block count will not be printed in the report, even if it did change as well. Ignore_list overrules report_attributes where they conflict.

Troubleshooting your config

Making a config file is a lot of hard work and must be done on a case by case bases. Don't give up simply because you don't get it right the first time around. This section gives you a few hints howto debug your config.

You can use `aide --verbose=255` to generate a lot of debug output to help you see which files get added and which are discarded. The following section gives some more information about AIDE's rule matching algorithm.

Understanding AIDE rule matching

Before reading this you should have basic understanding of how regular expressions work. There are several good books about this. Several Perl-books have also decent explanations about this subject. Just remember that Perl has some extensions to the standard regexps. There are also some differences in how different platforms handle regexps if you are using your platforms own regexp implementation. For example GNU regexps have their own extensions. Try reading the manual page of your system in this case. It might be a pain to read but it is worth it.

As you already know, aide has three types of selection lines:

- Regular selection lines, beginning with "/".
- Equals selection lines, beginning with "=".
- Negative selection lines, beginning with "!".

The string following the first character is taken as a regular expression matching to a complete filename, including the path. In a regular selection rule, the slash is included in the regular expression. An implicit ^ is added in front of each rule. A group definition follows the regular expression.

When reading the configuration file, aide internally builds a tree that roughly resembles the directory tree to be checked. Each node corresponds to a directory, and each node has one rule list for the associated regular selection lines, one for the associated negative selection lines and one for the associated equals selection lines. If there is no associated rule, the respective list may be empty.

aide tries to place a rule as far down in the tree as possible while still assuring that it is above all files that it matches. This is determined by the first "special" regexp character in the rule. For example, `!/proc` would be placed in the root node, `!/proc/.*` would be placed in the /proc node, `!/var/log/syslog*` is placed in the /var/log node and, finally, `!/home/[a-z0-9]+/.bashrc$` is placed in the /home node.

The algorithm that aide uses for rule matching is described in the following paragraphs. The pseudocode is an adaption from src/gen_list.c.

```
check_node_for_match(node,filename,first_time)
        if (first_time)
                check(equals list for this node)

        check(regular list for this node)

        if (node is not the root node)
                check_node_for_match(nodes parent,filename,false)

        if (this file is about to be added)
                check(negative list for this node)

        return (info about whether this file should be added or not and how)
```

When aide needs to determine whether a file found in the file system is to be checked, it first determines the deepest possible node x to match the current file against (that algorithm is not part of the pseudocode above), and then calls check-node_for_match(x, filename, true). So, the recursion starts at the deepest possible match.

As it can also be seen, equals selection lines are only checked in the first recursion step, thus providing some kind of speed optimization by reducing the number of necessary regular expression evaluations, which is a quite expensive operation.

Pitfalls

There are some side-effects from this algorithm that might seem strange at first. For example if you have the following rules:

```
/ R
=/etc R+a
!/etc/ppp/logs
```

The result would be that /etc and all files in it and in /etc/ppp except /etc/ppp/logs would be added to the database. This is perfectly normal. This happens because the =/etc matches not only /etc but all the files under it. Remember that regexps match always just the part they are referring to. The rest of the line is included by default. So `=/etc$ R+a` would be the correct form. If you don't have the `!/etc/ppp/logs` you would get the results that you are looking for because there is no node /etc in the regexp tree and there for it is not checked when AIDE constructs the list of files to add to the database. But when you have the negative rules the nodes /etc and /etc/ppp get created and they get checked when the file list is generated. So the =/etc is used to find a match in those nodes and it succeeds.

Consider the following rules:

```
/ R
=/var/log/messages$ R+a
!/var/log/messages.*
```

This is what you might write if you want to check /var/log/messages but not /var/log/messages.0 and /var/log/messages.1 etc. However since the negative selection rules are checked last and .* can match to an empty string /var/log /messages is not added to the database. The following is a more correct way of doing it.

```
/ R
=/var/log/messages$ R+a
!/var/log/messages\.[0-9]$
```

Now only messages files ending in number 0-9 and not included in the database. Note an intruder could disguise a rootkit by creating a directory called messages.9. If messages.9 does not already exist that is.

Consider the following rules:

```
/ n+p+l+i+u+g+s+b+m+c+md5+sha1+rmd160+haval+gost+crc32+tiger
/etc$ n+p+l+i+u+g
/etc/resolv.conf$ n+p+l+u+g
```

This way, changing /etc/resolv.conf will also report /etc as having their mtime and ctime changed, even if /etc is configured not to be checked for mtime and ctime. The reason is that aide only uses a deepest-match algorithm to find the tree node to search, but a first-match algorithm inside the node. Since /etc is in the / directory, /etc will match the rule for the root directory and ignore the specialized /etc rule.

Rearranging the configuration like this

```
/etc/resolv.conf$ n+p+l+u+g
/etc$ n+p+l+i+u+g
/ n+p+l+i+u+g+s+b+m+c+md5+sha1+rmd160+haval+gost+crc32+tiger
```

will solve the issue. It is generally a good idea to write the most general rules last.

Usage

First you must create a database against which future checks are performed. This should be done immediately after the operating system and applications have been installed, before the machine is plugged into a network. You can do this by giving the command `aide --init`. This creates a database that contains all of the files that you selected in your config file. The newly created database should now be moved to a secure location such as read-only media. You should also place the configuration file and the AIDE binary and preferably the manual pages and this manual on that media also. Please remember to edit the configuration file so that the input database is read from that read-only media. The config file should not be kept on the target machine. The attacker could read the config file and alter it or even if he does alter it he could place his rootkit to place that AIDE does not check. So the read-only media should be accessible only during the check.

Now you are all set to go. You can now check the integrity of the files. This can be done by giving the command `aide --check`. AIDE now reads the database and compares it to the files found on disk. AIDE may find changes in places that might not expect. For instance tty devices often change owners and permissions. You may want to read long reports and that is up to you to decide. But most of us do not have the time or the inclination read through tons of garbage every day. So you should trim the config file to include only the files and attributes of certain files that should not change. But keep in mind that you should not ignore too much as that leaves you open for an attack. An intruder might place his/her/its/their root kit in a directory that you have ignored completely. One good example is /var/spool/lp or something similar. This is the place that lp daemon stores its temporary files. You should not ignore it completely however. You should only ignore the format of files that you lp daemon keeps creating. And remember to use the $-sign at the end of your regexps. This stops someone from creating a directory that is ignored along with its contents.

Now that you have trimmed your config file you should update the database this can be done `aide --update` command. The update command also does the same thing as check but it also creates a new database. This database should now be placed on that read-only media along with the new config file. The check, trim, update cycle should be repeated as long as necessary. I recommend that the config file should be reviewed once in a while. The definition of "a while" depends on your paranoia. Some might want do it daily after each check. Some might want to do it weekly.

There is usually some drift in the databases. What I mean by drift is that new files are created, config files of applications are edited, tons of small changes pile up until the report becomes unreadable. This can be avoided by updating the database once in a while. I myself run the update every night. But, I don't replace the input database nearly as often. The replacement of the input datbase should always be a manual operation. This should not be automated.

There is also an alternative way of doing this. This method may be preferable for people that have lots of machines that run aide. You can run `aide --init` on all of the hosts and move the generated databases to a central host where you compare different versions of the databases with `aide --compare` This has the benefit of freeing up resources on the monitored machines.

Database and config signing

The security of AIDE can be increased by signing the configuration and/or database. When a database is signed, and it is changed manually, AIDE will refuse to use it. Likewise, if a configuration is signed, AIDE will not use it until the embedded hash is updated as well.

To make use of the signing features, use these options to the configure script:

--with-confighmactype=TYPE
: Hash type to use for checking config. Valid values are md5 and sha1.

--with-confighmackey=KEY
: HMAC hash key to use for checking config. Must be a base64 encoded byte stream. Maximum string length is 31 chars.

--with-dbhmactype=TYPE
: Hash type to use for checking db. Valid values are md5 and sha1.

--with-dbhmackey=KEY
: HMAC hash key to use for checking db. Must be a base64 encoded byte stream. Maximum string lentgth is 31 chars.

The base64 encoding was chosen so that the keys are not limited to printable characters. You can use a local base64 tool or an online base64 encoder to convert the keys to the right format. Then run configure, for example:

```
./configure --with-confighmactype=sha1 -with-confighmackey="YWlkZSBhaWRlIGFpZGUgYWlkZQo="
--with-dbhmactype=sha1 --with-dbhmackey="YWlkZSBhaWRlIGFpZGUgYWlkZQo="
```

To make the presence of a valid signature mandatory, the following configure options can be used:

--enable-forced_dbmd
    Forces the file/pipe database's to have checksum.
--enable-forced_configmd
    Forces the config to have checksum. Also disables --config-check

It is also possible to edit the `config.h` file by hand, and changing the values of the `FORCEDBMD` and `FORCECONFIGMD` macros.

Creating the hash for the aide.db database is done by running `aide --init` or `aide --update`. The hash for the aide.conf configuration file can be obtained by running `aide --config-check`:

```
$ aide --config-check
Config checked. Use the following to patch your config file.
0a1
> @@begin_config 27GF0+oKj1CvP4tltuibhu8YGIU=
13a15
> @@end_config
```

The `@@begin_config` and `@@end_config` can be added to the aide.conf file manually, or the output of `aide --config-check` can be directly piped into `patch`:

```
$ aide --config-check | patch
can't find file to patch at input line 2
Perhaps you should have used the -p or --strip option?
The text leading up to this was:
--------------------------
|Config checked. Use the following to patch your config file.
--------------------------
File to patch: /etc/aide.conf
patching file /etc/aide.conf
```

Using `forced_configmd` will make AIDE refuse to use unsigned configuration files. This also disables the `--config-check` option. This only makes sense if you already have a signed configuration, or if you have an AIDE executable on another machine that can create the signed configurations for you.

Miscellaneous

The AIDE database can be used to find the real names and places of files that have been moved to lost+found directory by fsck.

General guidelines for security

1. Do not assume anything

2. Trust no-one,nothing
3. Nothing is secure
4. Security is a trade-off with usability
5. Paranoia is your friend