ATP
Anti Tampering Program

David Vincenzetti
vince@dsi.unimi.it

Massimo Cotrozzi
cotrozzi@dsi.unimi.it

Computer Science Department
University of Milan – Italy

ABSTRACT

Since the earliest days of the *Unix*$^{(tm)}$ operating system, it was clear that, being impossible for a system administrator to monitor every single change in the file system, it would be a fertile playground for intruders. Every time intruders get into a Unix system, they tend to install a backdoor, just in case the hole they used to get in is fixed. Usually this is accomplished by modifying a system executable to contain code that allows him/her to access the system without authentication. This action is commonly called "file tampering" or "leaving a Trojan Horse". The program we present has the target to discover whether the integrity of a system has been compromised; in brief, we have developed a tool called ATP (Anti Tampering Program) which performs post-attack analysis.

1. INTRODUCTION

In recent years, attacks to computer systems have grown in number and in sophistication: in the early days the most common method of intrusion was password "guessing", whereas lately there has been a radical change in techniques used, and now some network attacks don't even require the intruder to have an account on the machine. Almost every machine which is on the internet have been "visited", or at least, "knocked on" by an intruder. This kind of situation is not avoidable as it is just like having a door on the street: sooner or later somebody will try to see if it is open or not. Bad guys exist in the computer community as in the "real world". Not only do they exist, but, in the majority of the cases, they spend most of their time studying new techniques for intrusion. Studies have been made on the aims of the bad guys, and it was pointed out that a tiny percentage of "hackers" hack systems in order to destroy information or for espionage purposes. Most of the cases, in fact, involve people just acting out of curiosity, or just to make the computer community know they are better programmers than system administrators.

The Internet worm, in 1988, exposed a set of classic operating system bugs but today it is not only a matter of operating system bugs anymore. Malicious hackers are learning how to communicate secretly using cryptography, how to foul up common Unix daemons by changing the source address of TCP/IP packets and by enabling IP source routing, how to penetrate into a system which is "firewalled" by Cisco router access lists only. Furthermore Unix systems are configured, by default, in a very open way and most system administrators don't care much about enhancing their system's security level. Being connected to the Internet we are potentially vulnerable to some new attack techniques of which we are not aware, so we can hardly ever state that our system is absolutely safe. The eventuality of a successful attack to our system is a real thing; in such an event we must be equal to the situation. When a Unix system is successfully attacked by an intruder and he/she gains root privileges it's up to him/her to decide whether to delete all system files or "just" to install a backdoor. If the intruder deletes files or destroys data it will be extremely painful to repair damages, but, on the other hand, the act of vandalism would be surely noticed by the sysadm. Otherwise if he/she deletes nothing but installs a backdoor, probably nobody would realize it and the system's security would be entirely compromised from then on.

ATP was created to solve this problem. ATP is intended to give system administrators a tool to protect the system from unwanted modification of sensitive files, i.e. configuration files, system daemons, most frequently used binaries etc. These files are, in fact, the ideal candidates for backdoors.

2. ATP – ANTI TAMPERING PROGRAM

ATP "takes a snapshot" of the system, assuming that you are in a trusted configuration, and performs a number of checks to monitor changes that might have been made to files. Of course it is useless if system integrity has already been compromised. In this case one should reinstall system binaries from scratch and reconfigure the system in a safe way.

To perform file analysis, ATP creates a database of file attributes. For every file in the database some `stat` informations, a double checksum and some flags are mantained. `Stat` informations consist of UID, GID, permissions, datestamps, size, number of links etc. The double checksum is composed by two parts: a CRC32 checksum and a MD5 checksum. CRC32 (32 bit Cyclic Redundandy Check) is a very popular algorithm and it is actually used in a wide variety of situations; CRC32 is very fast and reliable in detecting errors. Unfortunately CRC32 isn't considered a *secure* hashing function since it wouldn't be too hard for a malicious hacker to tamper with a file and then add some weigthed padding to the file in order to make CRC32 match. That's why ATP uses MD5 in conjunction with CRC32. MD5 is a RSA-DSI product and it is considered by many the state of the art of signature algorithms. MD5 produces a 128 bits message digest and it is conjectured that the difficulty of coming up with any message having a given message digest is on the order of 2^{128} operations ($\approx 10^{38}$). CRC32 is fast while MD5 is likely to be overkill for most systems, so the first one is intended for frequent massive checkings while the latter should be used from time to time, for example on a nigthly basis. Speed is not a significant parameter in an environment where you monitor a small number of files, but it is an important consideration in an environment composed of several workstation, with many NFS-mounted file systems and a very large number of files to be monitored. After "profiling" ATP at execution time, we discovered that more than 95% of time used was for calculating checksums. That's why ATP doesn't really use the original MD5 but a *modified* version of MD5. The changes, suggested by its author Ronald Rivest, have been made in order to speed up the checks, but the algorithm, according to Rivest, is still capable of providing a good level of security.

The whole database is encrypted using the DES cipher. In particular DES is used in Cipher Block Chaining (CBC) encryption mode with a pseudorandom Initialization Variable (IV). The database itself is signed with MD5 prior to encryption. A database checksum, a revision number and a timestamp are maintained to assure that any tampering with the database file would be discovered.

3. THE DATABASE

When the database file (tipically `/usr/local/lib/atp_database`) is created, a database encryption key is requested and this key will be used, from here on, for every access to the database. An alternative method of specifying the key is to set the environment variable ATP_KEYWORD to the desired password (however we must point out that on some operating systems this would be very dangerous! For example on SunOs anyone can display the environment for all users with the command `ps -eauxwww`).

The database is first built by taking files as command line parameters. For every other invocation of the program, the files specified are added to the database, if they are not already present, or replaced otherwise.

As files are registered in the database filenames are converted to absolute, machine dependent pathnames. Filenames are worked out by following all symbolic links and NFS mount points.

The database is an ASCII file. Lines in the database are composed by the following fields:

- The actual host the file phisycally resides on
- The real pathname of the file on that machine
- I-node of the file
- File permissions
- Number of links

- User ID
- Group ID
- Size in bytes
- Modification time
- Creation time
- Double file checksum

The database name can be specified at compile time, by defining the variable DATABASE in the Makefile, or it can be specified on the command line via the " -f " option.

Four more actions can be performed upon the database:

- Printing on the standard output (" -c " option)
- Deleting files from the database (" -d <file>" option)
- Destroying the database (" -D " option)
- Manual editing the database (" -E " option)

The last action is somewhat dangerous, for it is the only moment when the database will be written down on disk in an unencrypted way, via the temporary files created by `vi` (which is the default editor). Two definable environment variables are related to this option: EDITOR and TMPDIR, which define the path in which temporary files are to be created. If a malicious hacker modifies the temporary file during a manual editing of the database, modifications would be successfully added to the database, because after any "hand modification", the checksum of the database is obviously recomputed before the final encryption. This option is thus intended for exceptional management only.

4. SETUID FILES

Setuid files are a problem in almost every computing environment, so a specific option have been provided to handle this situation.

Specifying the " -S " option on the command line, all SUID files are searched and then inserted into the database; the method for doing this is by using the `find` command which searches throughout the file system for all "sensitive" setuid files; the program can be told whether a setuid file is sensitive or not. As an example we consider sensitive a setuid file owned by root or any other system related UID (i.e. `bin`, `adm`, `sys`, etc.) to be sensitive, but we allow *joeuser* to have any setuid file owned by himself or by any other user. By editing `atp.config` it's possibile to set `find`'s search criteria and to create a list of sensitive UIDs.

5. CHECKS AND ACTIONS PERFORMED

There are two ways to perform checks on files stored in the database: a basic one and a custom one.

5.1. Basic Operation Mode

The checks which are executed in basic mode are obviously based upon the information stored for each of the files; the most significant tests are:

- Comparison of the checksum
- Comparison of the permissions
- Comparison of the UID and GID
- Comparison of the creation and modification times
- Comparison of the number of links
- Comparison of the file size

Three kind of actions can be performed as a consequence of the positive result of any of the checks. These actions can be selected on the command line using parameters " -k1, -k2, -k3 " which define the three levels of response to a file tampering. All three checking options monitor, for every file in the database, the consistency of the parameters stored with the effective parameters taken from the disk. The default checksum algorithm is MD5. Alternatively you can specify, via the " -C " option, the utilization of CRC32 only. This is for speeding up the checking process just in case you are continously running ATP in checking mode (i.e. by `cron` once an hour).
If the " -k1 " option was specified, a report is written on standard output for use by the security administrator. By specifying the " -k2 " option you force ATP to take the following actions upon failing some of the checks: "zap" the file (that is `chmod 000` and `chown root` the file) and notify the security administrator (and/or anyone specified in the ATP_RECIPIENTS environment variable) via `mail` and `syslog`.
The " -k3 " option, finally, adds to the actions performed by the two previous options the search for illegal SUID files, that is files which resides in the file system but which are not in the list of "legal" ones (which are already in the list). This is checked with the same command `/bin/find` as above (redefinable run time by means of the ATP_FIND_COMMAND environment variable).

5.2. Basic Mode Examples

The following example shows how the basic features of ATP can be used.

```
$ atp
atp:  Bad options.  'atp -h' for a kinda help
$ atp -h
```

```
atp -- Anti Tampering Program: a file integrity checker for Unix
written by David Vincenzetti <vince@ghost.dsi.unimi.it>
Usage: atp [options] [files...]
possible options are:
        <files>...        :  Add/replace files in the database
        -f <file>         :  Specify database file
        -E                :  Edit the database directly
        -c                :  Display database to stdout
        -d <file>         :  Delete file from database
        -D                :  Destroy the database
        -x                :  Change encryption key
        -t                :  Edit configuration file
        -S                :  Add SUID files into the database
        -C                :  Use CRC32 instead of MD5
        -k{1, 2, 3, 4}    :  Check for file tampering
        -k1       :  Plain check, send report to standard output only
        -k2       :  Like above + zap illegal files, use mail and syslog
        -k3       :  Like above + search illegal SUID files
        -k4       :  Enable custom checking

Relevant environment variables:
ATP_KEYWORD, ATP_EDITOR, ATP_FIND_COMMAND, ATP_RECIPIENTS,
ATP_MAILER, ATP_GIDLIMIT, ATP_UIDLIMIT
```

So, if this is the first time we run ATP upon our files, we would probably like inserting all SUID files in ATP's database.

```
$ atp -S
I see you have created no database yet: creating a new one.

Setting database keyword. Think it carefully! Please don't choose
a short or obvious password since it may be vulnerable to
'guessing' attacks. Please use at least a dozen characters.

Enter keyword:
Enter it again:

Scanning disk using: /bin/find / -type f -print
inserted: /bin/df
inserted: /bin/newgrp
inserted: /bin/passwd
inserted: /bin/su
...
Database: 749 files; 77689 bytes
Database successfully saved.
```

Now we have control over all SUID files. Let's insert some more critical files in the database. As an example we select all "rc-files" and all files in the `/bin` directory.

```
$ atp /etc/*rc* /bin
Password:
VERIFIED database ``/usr/local/lib/atp_dabatase'' (77689 bytes)
         database MD5      :  b5e0046dd264aa70d5b72ef911ccb647
         last modification :  Aug 13 11:11:04 1993
         DES-CBC IV        :  40004c2840004c30

inserted:  /etc/auditrc
inserted:  /etc/bcheckrc
inserted:  /etc/brc
inserted:  /etc/d.cshrc
...
inserted:  /bin/ar
inserted:  /bin/as
inserted:  /bin/basename
...
replaced:  /bin/df
...
replaced:  /bin/newgrp
...
Database:  875 files; 90667 bytes
Database successfully saved.
```

Having inserted most of our critical files into ATP's database we can be pretty sure that any tampering will be detected. As an example let's suppose that an intruder illegally gained root privileges and he/she wants to install a backdoor on our system by changing the `/bin/login` program into a backdoor. This is what happens when we check things with the " -k1 " option:

```
$ atp -k1
Password:
VERIFIED database ``/usr/local/lib/atp_dabatase'' (90667 bytes)
         database MD5      :  b872f8772288b34a9a9da7bc7346f1e5
         last modification :  Aug 13 11:14:58 1993
         DES-CBC IV        :  40004c2840004c30

Comparing database with real files...

(1) FILE `ghost:/bin/login':
SIZE has changed from 159744 to 171008.
MTIME has changed from May 12 11:05:16 1992 to Aug 13 11:25:40 1993
CTIME has changed from May 12 11:05:16 1992 to Aug 13 11:25:40 1993
```

```
MD5 had changed from e13c9975dc636effc3c14ff9e83fce1a to
0ee985c2eccd6681866569e70411edef.

WARNING! File tampering detected!
Please pay careful attention to the above files because they have
changed since the last database update.  Someone might have
deliberately tampered with them.  Please check it out.
```

5.3. Custom Operation Mode

ATP's configuration file (`atp.config`) permits extremely flexible file specification and management. ATP can powerfully handle files from a very complex and heterogeneous environment. The checks to be performed can be selectively defined according to file's importance. The "actions" which are to be taken when a certain file appears tampered are fully configurable.
The only way to edit `atp.config` is by using the " -t " option. With the " -t " option an internal, self contained editor is invoked; this is needed since the config file is stored to disk in enciphered form and what's more we want no plaintext temporary files to be created. The encryption algorithm and the integrity checks used for the database are applied to the config file as well.
ATP's configuration file has the following form:

```
regexp      check-list      action-list
```

`Regexp` defines a *File Domain* (FD), that is a group of files for which the checks defined in the `check-list` field are to be performed. Upon a failure of any of these checks the actions defined in the `action-list` field are executed. The `regexp` field describes ATP's internal pathname representation which is composed by two parts: a hostname and a standard Unix absolute pathname. When using the " -k4 " option the files listed in the database are examined. For each file the rules specified in the config file are applied. The checks are performed on a first match basis, like Cisco router access lists. So, it's adviseable to keep a default rule in the config file in order to guarantee that all files are checked.

Actions can be `#defined` at the beginning of the configuration file. This comes handy since you can create your own checking macros. However one builtin macro is provided, BLTN_ACTION, which acts like " -k2 " in the basic checking mode.

5.4. Custom Mode Examples

Suppose our config file looks like this:

```
#define SNAPSHOT "/usr/local/bin/make_snapshot >> /usr/adm/atp_log"
# Checking /bin/login for all hosts.  Let's check for Checksum,
```

```
# creation time, mod time, size, link(s) number and perms changes.
# In case of unexpected modifications perform builtin action and
# take a ``snapshot'' of the system.
*:/bin/login              Ccmslp   BLTN_ACTION;SNAPSHOT
# Let's check all /bin system files for ghost.
# In case of unexpected modifications take the standard steps
# for sysadm notification and file zapping.
ghost:/bin/*              Ccmslp   BLTN_ACTION
# Checking RPC services for hosts hp1, hp2 and hp4.
hp[124]:/usr/etc/rpc.*    Ccmslp   BLTN_ACTION
# Checking logfiles for ghost.
ghost:/usr/adm/*log       lp       BLTN_ACTION
# Default rule.
*:*                       Ccsp     BLTN_ACTION
```

The following example shows how to use the " -k4 " option:

```
$ atp -k4
Password:
VERIFIED database ``/usr/local/lib/atp_dabatase'' (13905 bytes)
        database MD5      :  40a3ed879d5b1dddbc4e23480165243b
        last modification :  Aug 13 11:27:48 1993
        DES-CBC IV        :  40004c2840004c30

VERIFIED config-file ``/usr/local/lib/atp.config'' (1539 bytes)
        config-file MD5   :  bdef1d3d85c6696d7fd0f68e17c7f7de
        last modification :  Aug 11 22:49:08 1993
        DES-CBC IV        :  8473847384738383

Comparing database with real files, applying config rules...

(1) FILE `ghost:/bin/login':
SIZE has changed from 159744 to 171008.
MTIME has changed from May 12 11:05:16 1992 to Aug 13 11:25:40 1993
CTIME has changed from May 12 11:05:16 1992 to Aug 13 11:25:40 1993
MD5 had changed from e13c9975dc636effc3c14ff9e83fce1a to
0ee985c2eccd6681866569e70411edef.

Executing action: `/usr/local/bin/make_snapshot...'
Action return code: 0

(2) FILE `hp3:/bin/sh':
CTIME has changed from May 12 11:05:16 1992 to Aug 13 11:28:03 1993
```

```
PERMS have changed from -rwxr-xr-x to -rwxrwxrwx

WARNING! File tampering detected!
Please pay careful attention to the above files because they have
changed since last database update.  May be someone deliberately
tampered with them.  Please check it out.
```

6. CONCLUSIONS

ATP performs checks that can show with very high degree of confidence whether a file have been modified or not.

It is more powerful than other programs (such as Gene Kim's `Tripwire`) because of its sophisticated configuration language and its fully configurable set of constrains and actions for every single file, and it is more secure, for it is self contained and, with the exception of the " -E " option described above, it *never* writes to disk the content of the database in plain form. This is a very important point since the weakest part in Tripwire's security is the database integrity itself. In fact the first thing a malicious hacker does when he/she gains root privileges is to patch system binaries (such as `/bin/login`) and system daemons (such as `/usr/etc/in.remshd`) and alter Tripwire's database. It doesn't help much to protect the database by keeping it on a read-only mounted file system since the hacker can unmount it and mount it read-write. ATP doesn't require that the database be kept on a read-only media since the database cannot be meaningfully altered by anyone not knowing the right encryption key. Of course it would be better to backup ATP's database from time to time.

What's more, ATP provides a full mechanism for handling multi vendor NFS-mounted file systems (including HP-UX CDFs), and procedures that handle symbolic links through different filesystems. This is dramatically important in a distributed environment in which ATP can be executed from different clusters and files can have different names depending on the host we are accessing them from; in this case the program recognizes files which reside on a remote file system and if actions or checks are performed on such files, the same output is given.

Compared to ATP, Tripwire has two important drawbacks. The first is its incapability in following symbolic links. This is somehow surprising since files referred by symlinks are not registered in any way and this may lead to a false "sense of security". The second is the way it manages files from different hosts. Tripwire, in fact, keeps a different database for each system on which it runs. This works well when not using NFS but it becomes unacceptable otherwise. This is clearly shown in the following scenario. Suppose we have two systems, A and B, and we are using NFS for making A's `/usr` directory available to B. Alice, A's sysadm, thinks she'd better take care of, let's say, `/usr/etc/rpc.mountd`. So she inserts `/usr/etc/rpc.mountd` in Tripwire's database. Bob, B's sysadm, does the same. Then CERT announces and distributes a security patch about the mount daemon and Alice installs the patch on A. After installing the patch Alice updates Tripwire's database for host A. When Bob checks for the integrity of his system using B's database he will find out that `rpc.mountd` has changed and he will believe that someone has tampered with it!

7. FUTURE PLANS

In the near future we plan to design a method for handling largely distributed file systems: our effort will be in developing a sort of "atp server" called `atpd`, which should serve as a database centralization, to which requests should be directed for files which are not visible to the "local" file system. Of course all data transmission between master and clients would be encrypted; we are considering using public key encryption or some other method of cryptographic secret sharing.

8. AVAILABILITY

At present the program is in its final development state. The current version is 1.99. It's been tested on a variety of architectures such as HP, SunOS and SYSVR3. It will soon be available by anonymous ftp on `ghost.dsi.unimi.it` in the `/pub/security` directory.

For further info please contact David Vincenzetti, DSI – University of Milan, via Comelico 39, I-20135 Milan, Italy.

9. REFERENCES

[1] "Responding to computer security incidents: Guidlines for Incident Handling", Schultz, Brown, Longstaff University of California – 1990

[2] "Open Systems Security, an Architectural Framework", Arto Karila – 1991

[3] "RFC 1321: The MD5 Message-Digest Algorithm", R. Rivest, MIT Laboratory for Computer Science and RSA-DSI – 1992