Global Information Assurance Certification Paper

An Introduction to
The Coroners Toolkit.
Written By: Emil J.V. Björsell
17th January 2001

Table of Contents:

1.0 Introduction:

The Coroners Toolkit (from here known as 'TCT') is a suite of tools written for the purpose of gathering and analyzing forensic data typically from Unix systems. The toolkit was written by Dan Farmer and Wietse Venema, the people that brought you Satan, tcpwrappers. etc. The aim of this paper is to give you a brief introduction to the toolkit and some examples of it's application.

1.1 What is Computer Forensics?

Computer Forensics is pretty much like crime scene forensics. An incident occurs which must be solved. In order to solve the mystery we must set about gathering relevant information surrounding the scene, taking great care so as not to destroy our vital, and sometimes delicate evidence. After we have collected our evidence we must analyze the data and try to extract some meaningful clues that will help us establish what really took place and what caused it to take place.

Instead of collecting fiber samples, strands of hair, or finger prints we must try and gather clues from a computer system. We have to look at the disk drives, and memory, the operating system and it's files, the permissions and ownership of said files, and at what point did such things change.

1.2 The Coroners Toolkit

TCT is a set of tools that were developed by Wietse Venema and Dan Farmer in order to collect and analyze forensic data from a computer system. It is built up from two groups of tools, one of which is to gather data in an unobtrusive manner, and the other is to analyze and process said data. It can collect such data as host information, IP

addresses, and routing tables. It is also capable of taking a snap shot of the state of the file system, collecting such data as file permissions and ownership, MAC times [Appendix A 1.0], and cryptographic signatures of files. Additionally it can record what processes are running on the host and take copies of actual process memory. It can even retrieve files that have been deleted from the file system.

2.0 Let's Begin

In this section we will cover how to obtain, and compile the TCT package, as well as different approaches to installing it.

2.1 Obtaining the TCT package

You can obtain the latest version of TCT from both:
http://www.fish.com/forensics/
http://www.porcupine.org/forensics/
This introduction was written using tct-1.03, the current release at time of writing.

2.2 Compiling and Installing

Extract the package:
$ tar -xzvf tct-1.03.tar.gz
OR
$ gunzip tct-1.03.tar.gz | tar xvf -

Change your present working directory to the source distribution:
$ cd tct-1.03/

To compile simply type:
$ make

Ignore any warnings about missing files, unless 'make' failed with errors. You now have successfully compiled your toolkit.

Installing:

By default TCT lives in its distribution directory, I assume this is intentional so that we do not disturb the system that we are currently working on (assuming this is the actual system we wish to perform our post-mortem on!). If you wish you can move the contents of bin/ to somewhere in your $PATH (ex: /usr/local/bin), or you can store them on a floppy or cd-rom so you can easily take them to another system for analysis.

3.0 What next?

If we were performing a real post-mortem forensics analysis, ideally we should not compile and install TCT on the compromised host. Instead what would be ideal is to isolate the compromised host and take a ready made distribution of TCT on a floppy disk or CD-Rom, and do an analysis of the systems process information, network connections, and all other 'ephemeral' data. After we have done this we could either remount the hosts disks as read only, or better again, remove the disks from the host and remount them on a new machine where the TCT is already set up and ready to go.

In this paper we will outline examples that are being carried out on a live system (my workstation) and we suggest

anyone do the same for *testing*, in order to get familiar with the toolkit.

3.1 The Tools

In the distribution directory we have two directories that interest us for now: bin/ and conf/

In bin/ we have several commands, some of which we will talk about below. The conf/ directory contains configuration files for various tools

3.2 The Grave-robber

The 'grave-robber' is our main data collection tool, in fact it is not a single program but a perl script that calls several sub-commands and stores their output in a database. One has several command options that can specify what kind of information to collect, and where to put it. For the sake of brevity we will run it as default, which will collect system configuration files, inode data from unallocated areas of the file system, record stat() information on all files, record md5 checksums of all files, save open files that have since been unlinked, save certain files defined in the TCT configuration files, information from system utilities, record information from /dev, and gather trust/auth information such as ssh keys, .rhosts and host.equiv files amongst others.

The grave robber invokes the following programs

pcat Process CAT

In fact pcat is only invoked if the -p flag has been passed to grave-robber. pcat copies the process memory from a running program. This is useful if a program is running but its executable file has been removed. pcat can also be run independently of grave-robber. You must use pcat with extream caution, never use this tool for the first time in a production enviroment as it may crash your system!

ils Inode LS

ils by default lists the inode information of all deleted files on the specified device. This also can be run independently of the grave-robber.

icat Inode CAT

icat runs by default from grave-robber, its duty is to copy a file using its inode number. One must specify the device that the inode lives on, as two different disk drives can have the same inode numbers on the same machine. Again icat can be run independently from grave-robber.

shell commands

The grave-robber by default also invokes a series of, if present, commands to gather host and process information such as process lists (ps) open files, pipes and tcp/udp ports (lsof) and network information (ifconfig).

stat()

Last, yet very important, grave robber records stat() information on each file and directory. stat() is a function available in perl and C (and many more) that will provide meta data from the inode table. Performing stat() on a file or directory does not effect any data such as MAC times. It is very important that grave robber stat()'s files and

directories, before it opens or lists them, as this would change the access time of the directory (unless we mounted the disk read only!)

3.3 mactime

mactime is a reporting utility that will display the change in a file system between a specified time range. It is capable of reading its information from the data gathered by grave-robber or from a live file system.

3.4 unrm

unrm as its name implies 'Un-Removes' or rather restores deleted data from unallocated disk space. When a file is deleted in Unix it's file descriptor or 'filename' that we would see in our file system is unlinked from it's corresponding inode and data sectors on the disk. The actual data is not removed or destroyed until new data is written across the disk area that the previous file was stored.

unrm reads all unallocated disk data from a specified device and outputs it to standard output. For this reason one should never attempt to unrm data from a specified file system and write the output back to the same device. If you have a 10 Gb hard disk which has 6 Gb of data and you attempted to restore the remaining 4 Gb, you would overwrite the unallocated data with the restored data, and probably before you actually got to read the data! Later on we will demonstrate how we can safely restore some data on a typical Unix workstation for testing purposes.

3.5 lazarus

unrum merely writes our unallocated data to a local file, which, in such a state, is not much use to us. This is why we have lazarus. This program attempts to assemble unstructured data into something meaningful, and it does a pretty good job of it too! lazarus is also capable of identifying what type of files it has restored by looking at their contents. For instance, lazarus will inspect the first 10% of a file and if what it reads comprises of ASCII printable characters, it knows that this is a text file of some sort. If not it assumes that it has found a binary file and handles it accordingly. In addition to this it can look for common features to determine what type of binary or test file it really is. For instance, if it found an ASCII file and it's first line was '#!/bin/sh' there is a very good chance that this is a shell script. lazarus is capable of identifying a variety of files including passwd, mailq, C source, HTML and even sniffer files! We will demonstrate its output later.

4.0 Putting the tools to work

We have talked briefly about what computer forensics is, how to compile and install TCT, looked at all the different tools to see what they do, and now it is time to put them to use and see what they are really capable of!

4.1 Configuration

In TCT we have a few configuration files, which we will not talk about a whole lot unless necessary. However we are concerned with one thing. TCT uses an environment variable to tell the programs where to find their configuration files, normally if you run the programs from where they were compiled this is not a problem, however if you do move them you can set the $TCT_HOME environment variable to the location of the toolkit.

Example:
$ export TCT_HOME=/home/jev/tct-1.03

4.2 Gathering the data

Assuming we have the TCT_HOME environment variable set we will proceed!

```
# cd $TCT_HOME
# ./bin/grave-robber /
```

Now grave-robber will swing into action trying to determine what operating system it is inspecting, does the system support features such as /proc, and then it will process all files in the present working directory, all the directories in our $PATH and and directories listed in $TCT_HOME/conf/look@first. By 'processing' these directories grave-robber first does a stat() on the directories and records all the data it can such as owner, group, permissions, and MAC times. Then if it is dealing with a directory it will list the contents and continue processing the files and directories within. It will also take a md5 checksum of all files.

As this goes on grave-robber writes it's new found data to
$TCT_HOME/data/<YOUR_HOSTNAME>_YYYY_MM_DD_HH_mm_SS_GMT_OFFSET/

To make things easier for scripting and such, a symlink named after the hosts hostname will be created that points to the real directory.

Within the data subdirectory we have two files, 'body' and 'body.S'. The body file stores data on all files (see table 1.0)

Data	Value
md5 checksum (md5)	e9b9790ee1db50db35f66cb29850b9dc
file	/usr/sbin/arp
st_dev	160773
st_ino	1261845
st_mode	33133
st_ls	-r-xr-xr-x
st_nlink	1
st_uid	0
st_gid	0
st_rdev	5211408
st_size	10824
st_atime	979300539
st_mtime	975770132
st_ctime	975770132
st_blksize	8192
st_blocks	22

The body.S stores the information for all S[UG]ID files. This information is also kept in body, body.S is there for our convenience.

In the command_out/ directory we will find the output from all shell commands. For instance the output for dmesg can be found in $TCT_HOME/data/<YOUR_HOSTNAME>/command_out/dmesg. Also for every command output, file there is a corresponding file with the same name plus a suffix of '.md5'. For instance dmesg.md5. This file contains the md5 signature of the file and the corresponding filename along with absolute path name.

In 'conf_vault' we can find all configuration files that have been collected by grave-robber, such as boot scripts, syslog config files etc. We can set what type of config files are to be collected by editing the $conf_pattern variable in $TCT_HOME/conf/grave-robber.cf. This is made up of regular expressions, if a filename matches one of the expressions it is copied to the conf_vault/. grave-robber also writes a index.html file in the conf_vault/ directory so

we can easily navigate our collection of config files.

The trust/ directory contains all files that happen to match the file name contained in the @user_trust_files array. By default it picks up the following files and directory contents:
.forward
.rhosts
.*history*
.pgp/*
.ssh/*
.ssh*/*

We can inspect all the different files in the data/ directory, but to extract some valuable data from the body file, about files and when they changed, we can use the mactime tool. See the next section.

4.3 Interpreting all that data

It's all well and good to have all this information about every file on the entire system, but this still isn't of great use unless we organize the data so we can see what happened during a specific time frame, and in what order files were accessed, modified or deleted.

With the mactime tool we can specify a specific range in time and it can tell us all about what files changed during this time frame and even within what order. It can use our readily constructed database, courtesy of the grave-robber, or it can report information to us on the fly. We will work with the preconstructed database for this example.

In this example we are operating as root, because grave-robber creates it's files with 0600 permissions. Instead of changing the permissions or ownership we will just run mactime as root.

mactime 12/31/2000-01/01/2001

mactime takes dates in the format MM/DD/YYYY Where MM is Month, DD is Day of month, and YYYY is the year. By specifying two dates separated with a '-' mactime will return what file changes have happened during that period of time. If we do not specify a second date, mactime will tell us what files have changed since the first date and either present day or whenever our database was created.

There are several useful command line options, some of which I will briefly explain below.

4.4 Dis-information

Suppose you were a black-hat who wrote a root kit. One aim may be to stay covert, or not reveal what methods you use to compromise, Trojan, or tidy evidence from a host. The best solution is not to leave any evidence at all. Of course it is well know that this can be very difficult to do a perfect job, often the results can be too perfect. So perhaps we could try and disguise the fingerprint, as it were, of our tool. For instance, let's say we could identify the popular t0rnkit by looking at the mactimes of all files on a system being able to say that 'this is typical behavior of the t0rn kit'. If we wrote our root kit in a way so that it's final function was to touch and manipulate things like mactimes, and then erase as much evidence of the original tool as possible. This would throw off fellow investigators off the real cause, and may fool many into thinking they were dealing with "t0rn kit again!".

4.5 Resurrecting the dead from the magnetic world

As we briefly mentioned above, unrm reads unallocated data blocks from the specified device and writes them to a local file. This can and should not be done on the same hard-disk or partition in a real situation. However for testing

purposes, we can restore some data. Assuming you have a typical Unix file system set up, with each a seperate partition for / (root), /var, and say /usr, and that you have let's say 80 meg's of space in the /var partition and a good 2 Gb of free space in /usr.

If we copy a few files that are unique to us, a good example would be your cv, or some email from a friend. Copy these to /var, anywhere at all. Now delete them, find out what device is mounted to /var.

Example:

```
$ df
Filesystem  1K-blocks Used     Avail    Capacity Mounted on
/dev/ad0s1a 49583     36668    8949     80%      /
/dev/ad0s1f 28399720  13140074 12987669 50%      /usr
/dev/ad0s1e 19815     8697     9533     48%      /var
procfs      4         4        0        100%     /proc
```

That would be /dev/ad0s1f that is mounted to /var, Next we will suck up all the unallocated disk space from this device and write it to a local file.

First make sure we or on a partition that has sufficient space.
```
# pwd
/usr/home/jev
# unrm /dev/ad0s1e > var_undead
```

unrm sends all the data that it finds to stdo, so it is up to us to redirect it to a file.

Now we have got all unallocated data blocks from /var (or /dev/ad0s1e) in one single file. However there is still not much order or structure to the data we have collected. In order to examine and retrieve any useful data we can use lazarus.

4.6 Putting structure into that data

The lazarus tool will read the file produced by unrm and try to structure the data into logical meaningful files. It will also attempt to tell us what type of files it has recovered by looking at the contents for common features. For instance, if it finds a file that begins with:
```
root:$1$Yjf.tUxF$aSZx/fQX9aCQ118/ZN4Fs0:0:0::0:0:Charlie&:/root:/bin/csh
```
(ehh ehh, hmm, A entry from MY /etc/passwd file? ;)
It would assume that this is a passwd file!

lazarus can produce output in regular ASCII or colorful clickable html! We will work with the html output, as this is far easier and intuitive to navigate.

```
# lazarus -h var_undead
```

The -h flag tells lazarus to output html. Depending on the amount of unallocated data that is in our 'var_undead', file, this operation can take a very long time. lazarus isn't intended to be exceptionally fast, so be patient, it has far superior virtues.

When lazarus has completed you will have 3 new files and two new directories, var_undead.html, var_undead.menu.html, var_undead.frame.html, and www/ blocks. From here, the easiest thing to do is get your browser of choice and open 'var_undead.frame.html' Make sure you have permissions to view the files.

You will be presented with a legend that describes what each letter means, and a representation of how the data

blocks looked on your disk. By clicking on one of the letters, you can see what the actual block contained. For instance, if you see a series of 'Llll' and click on that link, you are likely to see some previously discarded logs! You can have endless hours of fun looking through your garbage!

Secure Deletion:

There are several free utilities available to securely delete data from your hardware, I expect that crackers and their toolkits will start doing a better job of destroying their tracks as time progresses, it is likely that this is allready happening.

5.0 Conclusion

In Forensics we should never rely 100% on one shred of information. What we need to do is gather all the information and correlate it to make a clear picture. I hope you enjoyed reading this paper, and that you learned as much as I did while writing it.

I would like to thank Dan Farmer and Wietse Venema for writing yet another superb toolkit for Information Security. I would also thank you the reader for taking the time to read this paper.

5.1 References

Peter Gutmann "Secure Deletion of Data from Magnetic and Solid-State Memory" July 22-25, 1996. URL: http://www.cs.auckland.ac.nz/%7Epgut001/pubs/secure_del.html (17 Jan 2001)

Belinda Brockman "A Forensic Argument for Network Time Synchronization" November 20, 2000 URL: http://www.sans.org/infosecFAQ/legal/time_sync.htm (17 Jan 2001)

Phrack Accident "Playing Hide and Seek, Unix style." Phrack Magazine Volume Four, Issue Forty-Three, File 14 of 27 URL http://phrack.infonexus.com/search.phtml?view&article=p43-14 (17 Jan 2001)

Dan Farmer and Wietse Venema "Forensic Computer Analysis: An Introduction" Reconstructing past events. Dr. Dobb's Journal September 2000 URL: http://www.ddj.com/articles/2000/0009/0009f/0009f.htm (17 Jan 2001)

Dan Farmer "What Are MACtimes?" Powerful tools for digital databases. Dr. Dobb's Journal October 2000 URL: http://www.ddj.com/articles/2000/0010/0010f/0010f.htm

"The stat() function" URL: http://www.delorie.com/gnu/docs/g77/g77_309.html