

THE CORONERS TOOLKIT (TCT) INSTALLATION & USE

Taken From the TCT Documentation (Farmer & Venema)

Introduction

TCT is a collection of tools oriented towards gathering or analyzing data on a UNIX system. There is no single task or ultimate goal that they are directed to, but if there was a theme it'd be the reconstruction of the past, determining as much as possible what happened with a static snapshot of a system. There are currently four major parts to TCT:

- grave-robber
- the C tools (ils, icat, pcat, file, etc.)
- unrm & lazarus
- mactime

The C tools are generally run by the grave-robber, but also can be used manually. The TCT IS A STATIC ANALYSIS TOOL. Care must be taken to isolate the system from outside forces while collecting data. At the end of the data collection all data should be moved to safety. At the very least MD5 or PGP signatures should be taken off line so that you can validate the results in the future. We further recommend that you use PGP to sign the data. To sign with PGP, you generally do: ***pgp -bs foo.tar.gz***

The resulting signature file is: ***foo.tar.gz.sig***.

To verify, you can: ***pgp foo.tar.gz***

Installation

TCT requires that the system have a working C compiler and perl, version 5.0004 or later. The Isof utility is HIGHLY recommended. TCT is meant to be run from its own directory. Efforts to move individual files to various locations (/usr/local/bin or wherever) is left as an exercise for the reader. If you do so, be aware that the file command has a non-standard path for the magic file wired into it.

Installation is therefore simple. Find a location that has room, preferably lots of room, as several of the tools in the toolkit can generate lots of data and untar the thing.

The Makefile should automatically do the right thing. Simply type: ***make***

At this point, if all has gone well, you can start using the toolkit. Most of the useful tools/binaries are in the "bin" directory. For security reasons TCT uses full path names. If you're reinstalling it or have moved it around you'll want to run the "reconfig" program again. Simply type: ***perl reconfig***

Grave-Robber

The "grave-robber" tool is at the heart of much of TCT. It is little more than a data capturing tool, mostly running various commands in an attempt to gather the basic system information and to save some of the files on the system that are necessary to analyze it.

THE CORONERS TOOLKIT (TCT) INSTALLATION & USE

Taken From the TCT Documentation (Farmer & Venema)

For most investigations simply “getting” the raw, pristine data in a somewhat timely manner is more than the battle. In our own efforts we've found that automation is simply the only way to reliably capture all the necessary information.

TCT attempts to capture as much data as is both useful and pragmatic. There are two log files of general interest that are generated by the “grave-robber”: coroner.log and error.log. The first logs all the commands and when they were executed, the error log lists all the output going to stderr, which is often, but not always, of some interest. However, one should note, due to the extreme complexities of modern systems, for the most part, except for MAC times and unallocated data, little effort has been made to “interpret” data, simply collect it.

The “grave robber” is most useful when run over an entire system, capturing as much information as possible. Point it at the root file system if space permits. And, while TCT doesn't need to be run as root, running it from an unprivileged account will prevent you from capturing files and processes that the account doesn't own. You may or may not wish to run the grave-robber with the -p flag as this grabs all the process memory, which can crash a system. Test it out before using it in a real emergency

The last thing the grave-robber does is create MD5 signatures of all the output that was created during its latest run, in a file named "data/hostname/MD5_all". If you can't take all the output of TCT off the system (you should always try this!) it would be a good idea to at least keep the MD5 output off-line so the results can be validated later.

The grave-robber tool is less of a tool itself than a framework for other tools to reside in. In the current incarnation it runs a variety of perl modules, nearly all of them living in the "lib" subdirectory. It was explicitly designed to never send commands directly to the Unix command shell for parsing (to avoid nasty side effects of meta characters, among other things) and it also logs all shell commands and what time they were run at. It has a slew of options for running various commands, saving data in various locations, and other miscellanea.

It roughly runs these tools so that data is captured according to the Order of Volatility. The OOV roughly says that certain data is more volatile or ephemeral than other types. Generally speaking you want to capture the most volatile information before it goes away. However, since any queries of the system risk disturbing other potentially valuable data one must be careful. And while it impossible to automate this perfectly, the grave-robber can be a useful way of automating the process.

Before it starts doing much data collection, however, it examines some of the tools and files that are important to the system that you might wish to examine manually before waiting the long time it takes for the grave-robber to finish its run. Usually this includes files located in the /etc directory, but also might include commonly runs commands, such as those found in /bin, /usr/local/bin, etc. The configuration file look@first is found in the "conf" subdirectory and controls this.

THE CORONERS TOOLKIT (TCT) INSTALLATION & USE

Taken From the TCT Documentation (Farmer & Venema)

Often you'll want to run the grave-robber on the entire system, and by default it does just this. It is recommended that you run the grave-robber as root so that it can capture files and process information that are not available to normal users. To run it, simply type "grave-robber" (or "bin/grave-robber", if TCT's "bin" directory is not in your path).

By default, the grave-robber captures ephemeral information (process and network state), figures out what it can about the system hardware configuration (especially, disks and disk partitions), and searches the file system for critical files (configuration files, log files, other critical files). However, the grave-robber's file system scan can take several hours. All that processing is happening with the one and only original copy of the data. This goes somewhat against a basic rule of computer forensics analysis: do as little processing as possible with the original.

If you have the luxury of being able to "dd" the disks and of doing an off-line file system analysis, you will want to run grave-robber with no more than the -f flag (the "fast scan". This turns off the file system walk, and allows you to run the full-blown set of tools at your leisure on another system against a copy of the data.

With the exception of the error and command logs, all captured data is saved in the data subdirectory. To be precise it is actually kept in a subdirectory of the data subdirectory, the name of the subdirectory corresponding to the name of the system and the time that the grave-robber tool has been run. The grave-robber also creates a symbolic link, "data/hostname", to the actual data dir. In the hostname subdirectory there are several files and subdirectories of interest:

- **command_out** - A subdirectory that keeps the output of most of the programs that execute code under the grave-robber. Each file in this directory is generally named after the command run and its arguments. In addition an MD5 (of the output file) & time stamp (of when the program was run) is created, saved to the same filename with the ".md5" extension.
- **strings_log** - This is the output of the strings command on every directory the file walker found. Doing this will often reveal names of deleted files.
- **Body** - this is the mactime database.
- **body.S** - This contains file attributes of all SUID files. They are in the normal mactime DB as well, it's simply simpler to see them here.
- **coroner.log** - This contains the date and time of all programs run by the grave-robber program. This is in the main directory.
- **error.log** - This contains all errors generated by the grave-robber program. This is in the main directory.

THE CORONERS TOOLKIT (TCT) INSTALLATION & USE

Taken From the TCT Documentation (Farmer & Venema)

- **deleted_files** - This directory contains all deleted files that were still open or running when the grave-robber ran. (Done by a combination of ils & icat).
- **pcat** - This directory contains all the images of running processes that pcat recovered. Lots of interesting things can be found here! Among other things user shells often have their history in memory, some programs keep IP #'s or hostnames, etc.; a command such as (in SunOS the -s flag pretty prints od output in a nice text form, YMMV): `od -s pid.out.date | less` can tell you quite a bit about what was going on in the past!
- **conf_vault** - This directory is an archive of all the files that the grave-robber found of interest. Configuration files, critical files & directories, etc. Mostly controlled by the configuration files: `save_these_files`, `coroner.cf`, `grave-robber.cf`. In addition there is an `index.html` file that points (via HTML) to all the files in this subdirectory.

The grave-robber has a few command line options (see the man page), but mostly it is controlled by either editing the grave-robber file and removing or commenting out lines that you do not wish run or by editing the main configuration file, "grave-robber.cf" (or the global configuration file, "coroner.cf"). Most of the options in the configuration file do not need to be changed, and should be fairly well commented on in the configuration file itself.

MAC Times

Mtimes, atimes, and ctimes (or MAC times) are properties of a file that, if not intentionally modified or obfuscated, combine to be one of the most useful tools in forensic analysis. The mtime contains the last time the file was modified, the atime holds the last access time (for reading or execution), and the ctime is the last time the file status has been changed. Status is the meta data that is typically changed by a `chown`, `chmod`, etc. command.

Unfortunately MAC times are easily modified and are extremely ephemeral. The next time someone accesses or modifies a file in some way the prior values are lost, so haste must be exercised when collecting this information. However, if you are reasonably certain that the MAC times have not been distorted in some way, they are perhaps the surest way to find out what has been happening on a system.

Although the "mactime" tool can obtain MACtimes directly it doesn't save the results in a database, so it is usually desirable to use the "grave-robber", a program that lstats an arbitrary amount of the file system and saves the results for later processing. To run it you simply type: **`./grave-robber -m /directory-tree`**

The -m flag means only do MACtimes. Obviously, if you specify the root dir ("/") it should go through the entire file system. NFS directories are searched as well. Try it out on something like `/tmp` or `/bin`. Grave-rob should not disturb the system in any way.

Once the grave-robbing is done, you'll want to use the "mactime" program. Mactime is easy to use in its simplest form. Just type: **`./mactime 4/5/2000`**

THE CORONERS TOOLKIT (TCT) INSTALLATION & USE

Taken From the TCT Documentation (Farmer & Venema)

It will dump to stdout all the files that had their MAC times changed since then. Try it out! It's trivial to see what someone did on a system if you get to it quick enough, and even events that happened far in the past can be examined if they are unusual enough. Some really cool uses for this tool:

- Find out what files are touched/run during a system boot - esp. cool when comparing OS's.
- Nuking SUID/SGID files on a firewall
- Determining activity during a day or slice of time
- Finding out how much complexity (in terms of files) a windowing system really adds.

I've added some small HTML support - mild highlighting, etc. To use this, simply check out the options -u & -f. Example usage: **mactime -u root -f conf/mac_file.lst 5/1/2000**

Unrm & Lazarus

Many people think that destroyed or lost data, typically thought to be lost in Unix systems, can essentially never be recovered, either in whole or in part. We are here to tell you something different. Unrm and Lazarus work together for the forces of sloth and extreme wasting of disk space. Unrm is a C program that is essentially an electronic dumpster diving tool - it roots out all information on a Unix file system that is not currently allocated.

That means if you have a 10 GB disk and you're currently using 2, unrm would generate 8 gigabytes of data. Since UNIX tends to allocate file system blocks for different files in different places you must save this resulting data on a separate disk or you will destroy (overwrite) your potential forensic data. The unrm program typically requires root privileges to use.

Lazarus takes the data from unrm (or, actually, from any data source, such as RAM, swap space, etc.) and attempts to bring some order into what is generally perceived of as garbage. Perhaps it is most useful when the HTML output option is used and a click-able map is generated that allows browsing of the interpreted data.

Lazarus is a program that attempts to resurrect deleted files or data from raw data, most often the unallocated portions of a Unix file system, but it can be used on any data, such as system memory, swap, etc. It has two basic logical pieces: one that grabs input from a source and another that dissects, analyzes, and reports on its findings.

It can be used for recovering lost data and files, as a tool for better understanding how a Unix system works, investigate/spy on system and user activity, etc.

THE CORONERS TOOLKIT (TCT) INSTALLATION & USE

Taken From the TCT Documentation (Farmer & Venema)

While something like dd may be used for supplying input, perhaps the most interesting thing to do is to utilize the unrm program to obtain the currently unused space in a file system to analyze information that has previously been allocated and then released.

Lazarus (not unrm) has been used with data from UFS, EXT2, NTFS, and FAT32 file systems, but it can be used on just about any type of file system, your success will vary with the way the data resides on the disk, but it always seems to find something.

The dissection and analysis side of Lazarus is a perl program which takes the following steps:

1. Read in a chunk of data (typically 1K, but modifiable by changing the variable \$BLOCK_SIZE in lazarus.cf).
2. Roughly determine what sort of data it is - text or binary, so that further analysis can be done. This is currently done by examining the first 10% of \$BLOCK_SIZE, if they are all printable characters then it's flagged as a text block, else it is flagged as binary data.
3. If text, it checks the data against a set of regular expressions to attempt to determine what it is.
4. If binary, the file(1) command is run over the chunk of data. If it doesn't succeed, the first few bytes are examined to see if it appears to be in ELF format. Due to the bugs and problems with various versions of file we've ported the Free-BSD version to a few UNIX and include it in this distribution.
5. If it is recognized, the block is marked as a block of that data type. If it is a different type of block than the previous block then it is saved to a new file. If the data is the same type as the previous data block it is concatenated to it. If the data block is not recognized after the initial text/binary recognition but follows a recognized chunk of text/binary data (respectively), lazarus assumes that it is a continuation of the previous data and will also concatenate it to the previous data block.
6. The output is in two forms, the data blocks and a map that corresponds to the blocks. The blocks are saved in a separate directory (by default, amazingly, "blocks") in a file that starts with the logical block # that is currently being processed. Each run of similar data, having all been saved to a single file, also has a name that corresponds to its data type ("c" == C source code, "h" == HTML, etc.) Since data files might be viewed in a browser (more on that later) they all have an ending of ".txt" so that they will not be interpreted as potentially harmful code. The naming convention is blocknumber.type.txt, or blocknumber.type.{jpg,gif,etc} if it is an image file, so, for instance, a run of mail data might have the name 100.m.txt. So searching all the blocks identified as C code to see if they contain a header file, for instance, is as easy as something like: `grep -l header.h blocks/*.c.txt`

THE CORONERS TOOLKIT (TCT) INSTALLATION & USE

Taken From the TCT Documentation (Farmer & Venema)

The data map generated is an ASCII list of characters that corresponds to the various data types found. The first character that represents a "run" of blocks is capitalized, so that a run of mail would show up as: Mmmmm

To try to make the output stay within a semi-manageable size it does a logarithmic compression of the output (base 2) - e.g. the first character represents one block or less of data, the 2nd from 0-2 blocks, the 3rd 0-4, the fourth 0-8, etc., so that the above run of mail data (of 5 characters) would be 1+2+4+8 plus the last block, which could be from 1-16 K, totally 16-31K of data.

This is all most colorful (readable?) if viewed with a browser; it outputs colored characters (and a map legend), and clicking on the first character of a data block will display in a window with both a simple navigation bar and the data contained in the run.

There is an alternate method of analysis which does byte-by-byte examination, rather than a block at a time; it is significantly slower (and Lazarus is already very slow) but might be better suited for non-block based data (such as memory or unknown data fragments.) It basically is the same, looks at \$BLOCK_SIZE data sized chunks - but instead of looking at only the first 10% of the block it examines the whole thing looking for fragments.

After the initial analysis you essentially have one of two choices: use the output/data map to examine the data or go straight to the data blocks for further processing/searching.

Rare is the user who hasn't blown away data inadvertently. Most break-ins involved at least small amounts of data destruction (if nothing else intruders will often carry around or install the tools that they use to compromise systems); legitimate usage of a system - mail, WWW browsing, compiling programs, etc. leaves considerable amounts of deleted activity on the disk, etc. There are many reasons, some legitimate, some not, why a user might want to examine the system.

For better or for worse, there are several significant obstacles to doing data recovery or analysis on a Unix file system (which will probably be the most significant usage of Lazarus.) First and foremost, unlike PC's Mac's and other systems, most Unix file systems were designed for high performance with no (or not much) thought to data recovery. When a file is removed essentially all useful information about it - the filename, inode, etc. are either deleted or mostly rendered useless for data recovery.

So in order to do any sort of recovery you generally have to examine ALL the unused portions of the disk, which can take an enormous amount of time depending on how much free space you have. In addition, while (especially for smaller files) Unix generally attempts to write data in contiguous blocks, the larger the file the better the chance it has been broken up into pieces. While it is possible to manually reconstruct data in such a situation, it will probably be very painful unless the format of the data is very regular and easy to recognize, like mail, system log files, etc.

THE CORONERS TOOLKIT (TCT) INSTALLATION & USE

Taken From the TCT Documentation (Farmer & Venema)

Also, unless the disk is frozen, you run the risk of overwriting the data in question (and even if done immediately, kernel data buffers and the like could still be waiting to be sync'd. As a note - on the good side of things we've found that data is very localized; reasonably sized files in a directory will usually have all their data in the immediate neighborhood of that directory, so unless you're writing to that same area you're probably OK. Indeed, unless you really hammer the disk with writing, most stuff is probably going to be untouched, even long after the data has been deleted.

All said, however, there are probably two main uses for such a tool: data recovery and (often post-mortem) analysis.

Data Recovery

Hopefully you've only blown away a small easily recognized text file. This is probably not the case. Regardless of what happened, you'll want the following items:

1. A second system that can recognize the disk is optional, but desirable.
2. Another disk, or at least another file system on the same disk if you've taking data from one file system and writing it to another. This may sound foolish, but **UNDER NO CIRCUMSTANCES SHOULD YOU RECOVER DATA ON THE SAME FILE SYSTEM IT WAS LOST ON!!!!** If this isn't clear to you, consider; your data is in that free area out there somewhere. You'd be filling up that free space with itself, essentially at random locations. There'd be a more than significant chance you'd blow away your potentially interesting data before you got it.
3. At least as much free disk space on a **TARGET** location as the free space on the afflicted disk. Ideally you want a bit more than twice as much space, to both write the unallocated space and to save the lazarus results. This is because lazarus rewrites all the data in another location as well. So if you have a 2GB disk with 750MB free, you'd want a second disk with 750MB x 2, or 1.5 GB free. At least; if you try to reassemble the data in various ways you'll want at least a bit more space to play with.
4. Ample amounts of free time.

Then take the following steps:

1. Remove the disk from operational hazards. If it's a system disk that sees lots of action, halt the system and mount the disk on another system. Mounting it read-only is a fine idea, so that no additional data is lost.
2. Run unrm or optionally you can simply use dd. For instance:

```
# dd bs=1000k if=/dev/rsd0b of=whatever  
# dd bs=1000k if=/dev/rdisk/c0t3d0s1 of=whatever
```

THE CORONERS TOOLKIT (TCT) INSTALLATION & USE

Taken From the TCT Documentation (Farmer & Venema)

```
# ./unrm /dev/rsd0a > output  
# ./unrm /dev/rdisk/c0t3d0s0 > output
```

Of course dd will make no distinction between free and used blocks. If you're interested in analyzing the free space, use unrm if at all possible. If you're feeling brave you can try it on kmem/mem/swap/whatever, although don't blame us if your system doesn't like having it's memory dumped out (worked fine in our systems, but this is potentially dangerous stuff) and crashes:

```
# dd bs=1000k if=/dev/kmem of=whatever
```

3. Run Lazarus.

4. Now begins the interesting part. First, what sort of data was lost? If it was mail or other text files, you might be in luck. You can try to run the "rip-mail" program and see if it recovers the info. If it was a piece of text, writing, or mail that the rip-mail program didn't recover, then grep is probably your next best friend. Remember, all the blocks are saved in individual files in the blocks subdirectory. So, for instance, if you're looking for your resume, think of keywords that might help you out. You could then do something like:

```
% egrep -l 'keyword1|keyword2|...' blocks/*.txt > allfiles
```

If there are any files listed, start examining them with a good pager.

Images are likewise easy; simply do something like:

```
% xv blocks/*.gif blocks/*.jpg
```

Text based log files, even though they will often be spread out all over the disk because of the way they are written, are actually trivial to recover, and in the correct order, because of the wonderful time stamp on each line; the simplest way is to tr is in there to remove nulls:

```
% cat blocks/*.l.txt | tr -d '\000' | sort -u > all-logs
```

And then browse through them. A few bits and pieces will probably be lost due to the fragmentation at block and fragment boundaries, but it's a good way to start.

Lazarus should process and emit all data fed into it as input into the "blocks" subdirectory. If you concatenate all the output blocks and compare them to the original image they should be identical. It doesn't change the input at all, it simply breaks it up into more readable pieces.

While text output files created by lazarus can be as short as 1K, the minimum size for most binary files, irrespective of size will be the minimum block size that the disk can write. This is because after the first 1K is read in a text file will end if it hits binary information (e.g. the

THE CORONERS TOOLKIT (TCT) INSTALLATION & USE

Taken From the TCT Documentation (Farmer & Venema)

garbage at the end of a disk block). Binary files, however, will simply concatenate the garbage on the end of the recognized binary unless by some miracle it's recognized as another binary type. This could be fixed if lazarus actually looked at the binary format it was trying to read and found the size of the file, it could stop reading after that many bytes, and do better overall recognition.

If that's not clear, change the variable `$BLOCK_SIZE` in the lazarus configuration file "conf/lazarus.cf" to be a more reasonable value for the system you're investigating. 1024 is a very safe, but conservative number, but if you'd like to cheat smartly and increase performance set the block size 8192, which is the FFS logical block size. You'll then miss all the small files that may not start on a 8kb boundary, however.