Global Information Assurance Certification Paper

Installing, Configuring, and Testing The Deception Tool Kit on Mac OS X

By Jon Lucenius (GSEC Practical Exam - v1.4b)

Abstract / Summary

This paper will introduce a Honey Pot known as the Deception Tool Kit (DTK) written by Fred Cohen. It will give an overview of what the DTK is, where to obtain it, how it works, and offers advice about when it should be deployed. Out of the box, the DTK is readily installable on most Unix-based operating systems including Linux, but has no installation support for Apple's new operating system Mac OS X (OSX). For more information on OS X see Apple's website at http://www.apple.com/macosx/ (Apple)

The pre-requisites and changes that are necessary to install and run the DTK on OSX will be outlined for the reader, showing how it differs from a standard Unix system in the context of using this product. The goal of this paper is to facilitate the installation of the DTK by a novice user onto any Mac OS X machine. After it is installed, we will set-up the DTK to run on port 8080 and provide a deception on that port in response to a threat. After the set-up is complete we will test our deception port for the appropriate responses.

Conventions

- Commands - **`BOLD, FIXED-WIDTH FONT`**
- Responses - `NORMAL FIXED-WIDTH FONT`.
- My Comments – **`[NORMAL FIXED-WIDTH FONT]`**.
- Menu selections "`File -> Open -> Action Item`".
- Local Paths "`/Applications/Utilities/AppName`".

Overview of the DTK

What it is

The Deception Tool Kit is a Honey Pot. "Honeypots are unique technological systems specifically designed to be probed, attacked, or compromised by an online hacker" (Spitzer, Back Cover). The DTK is capable of simulating a wide variety of services on a system, and is capable of masquerading as several different hosts as well, giving it the capabilities of being a Honey Net. An excellent resource on Honey Pots, written by David Klug, is located in the SANS reading room at http://www.sans.org/rr/intrusion/honeypots.php (Klug).

An important component of the DTK is a psychological one. If there are enough people simulating enough services on enough machines, just the mere presence of the DTK on its own special pre-defined port 365 might be enough to convince the attacker that they are wasting their time, which in itself is a measure of defense.

The author, Fred Cohen, writes about this on the DTK website at http://www.all.net/dtk/dtk.html

> If the DTK becomes very widespread, one of DTK's key deceptions will become very effective. This deception is port 365 - which we have staked a claim for as the deception port. Port 365 indicates whether the machine you are attempting to connect to is running a deception defense. Naturally, attackers who wish to avoid deceptive defenses will check there first, and eventually, simply running the deceptive defense notifier will be adequate to eliminate many of the attackers. (Cohen - http://www.all.net/dtk/dtk.html)

Where to obtain it

To support the goal of making the DTK widely deployable it is free and publicly available from the all.net website. The homepage of the DTK is located at http://www.all.net/dtk/dtk.html . On the site you will find the following pertinent links

- **The DTK tar file** - http://www.all.net/dtk/dtk.tar.
- **Installation Instructions** - http://www.all.net/dtk/download.html
- **FAQ page** - http://www.all.net/dtk/faq.html
- **Example Deception** - http://www.all.net/dtk/example.html
- **Example Log File** - http://www.all.net/dtk/detections.html

How it works

Technically, the DTK is made up of C programs, Perl scripts, and additions/modifications to host configuration files that direct the attacker into the DTK, simulate responses, and record requests made of the host system. When a request comes in thru the wire that is directed to the DTK, the attacker enters a user defined **State Machine**. When the attacker issues a command or request it results in a pre-defined response, either encouraging the attacker to continue his exploration of the host or resulting in a shutting down of the service. The DTK is not expected to fool experienced attackers for a long period of time, but rather be enough of a deception to make the attacker think twice about launching further attacks on that host or others on that network.

The FAQ page contains the following exchange from an email:

> [un-named user]
> > DTK has patterns which can be detected. E.g. too much buggy software

> apparently running on a host whose sysadmin is otherwise clueful will
> be suspicious.

[FRED COHEN]
Indeed, but by the time you can tell the difference, we have recorded your illicit entry and, optionally, reported your activities to the systems admin. But even more importantly, if you back off when you find DTK running, then DTK has done its job. There's no getting around the fact that in order to find out if the defense is there, you have to trigger the defense, and by that time it is too late. (Cohen, http://www.all.net/dtk/faq.html)

When it should it be deployed

Honey Pots in general, and the DTK in particular, are useful in several situations. It can be part of a planned network, luring explorers of the network into a false sense of security. In this case, you could start by running the DTK on ports that run programs containing known security flaws, expecting that attackers would try to exploit those. Another useful scenario is when an attacker is actively exploring hosts on a network, and you have specific information about the ports that are being attacked and the responses that are expected in return. This could be a known security flaw that the attacker is attempting to exploit, or a Trojan back-door that they believe to exist and are trying to take advantage of in a pre-defined manner. It was for this reason that I originally installed the DTK. The third case is when a machine has already been compromised. Putting a Honey pot in its place, which can be the same machine with additional detection tools, can lure the attacker into a false sense of security and expose their methods.

How does the DTK compare to a commercial Honey Pot?

Some of the benefits of using the DTK include it being publicly available, free software that is installable on virtually any Unix based system including Solaris, Linux, and now Apple's in newest operating system OSX among others. It requires no special tools to set up and is able to perform its task well out of the box with little effort. However, the DTK does have some drawbacks. It will not fool experienced attackers for very long, each response must be pre-defined in the configuration file (which had no included GUI interface as of this writing), and the tool does not learn over time about an attacker's intents.

Some commercial software packages, such as ManTrap, initially by Recourse Technologies, now available thru Symantec, will do things such as update e-mail messages, change recent files, and respond in other ways like a real system over a given period of time. However, these software systems are relatively expensive and some must be maintained on a on-going basis to keep up the charade.

The following quote, written a few years back by Jeff Forristal, was taken from an article comparing the DTK to commercial Honey Pots.

> However, it [DTK] comes up short in some areas. It's possible to remotely determine that some of the services are emulated, and there is no consistency across the services as to what OS they are trying to represent. Like BackOfficer Friendly, DTK does not do any network-level OS emulation, so an attacker can use an OS identification tool, such as Nmap, to determine the actual platform, which may ruin the facade the honey pot is trying to present and otherwise alert the intruder. (Forristal – p2)

With regard to the above statement, consider that I have a deception port running on a PRODUCTION web-server. The server is running Windows 2000, and I have decided to emulate a Solaris 2.7 response via an ftp deception on port 21. You, the attacker, decide to 'confirm' that the OS is really Solaris. You 'discover' that is Windows 2000. You might conclude that either my server is really hosed, or that you have landed on a honey pot. You know you cannot trust the FTP port, and you can never be sure about the web server either, even if you can prove it is on Win2K. This is an example of the psychological aspect of the DTK that is so compelling. As a further measure, I can have a web server running on my sendmail machine as an additional deception, with the deception port 365 activated on only 1 of those machines. It is easy to see how these measures can waste an attackers time and resources. Even more important, thanks to deception techniques, they can never be sure if any server they attack in my network is real or a honey pot.

Pre-requisites for Installation

GET PERMISSION BEFORE STARTING - I cannot emphasize enough that you must get permission to install, configure, test and run this and most any information security software. This holds true even on machines under your direct control. Be certain to get the permission in writing (including keeping a hard-copy of any emails that you send and receive) and make sure it is from somebody who has the authority to make the decision and is willing to back you up if the need arises.

ADD DTK TO SECURITY POLICY - You should consider having the DTK added to one of your Security Policies. This can be a valuable aid in transferring knowledge to co-workers and subordinates, and can help in demonstrating to superiors where your time and efforts are being spent. Some careful forethought and planning will go a long way in this regard.

TCP Wrappers

Mac OS X comes with Wietse Venema's TCP Wrappers pre-installed. Although TCP Wrappers is already running, it is completely un-configured and provides no protection. For more information regarding TCP Wrappers see the CIAC page at (http://ciac.llnl.gov/ciac/ToolsUnixNetSec.html#Tcpwrappers)

or read the man page.

```
$ man hosts_access
```

Perl 5

Mac OS X also comes with Perl 5.6.0 pre-installed. A reference to the Perl binary is located at the standard perl location on a Unix platform at /usr/bin/perl. The actual Perl code is located at /System/Library/Perl. You can read the man page for more information about Perl.

```
$ man perl
```

The following prerequisites are specific for installing the DTK on Mac OS X version 10.2.x (also known as Jaguar).

Mac OS X Developer Tools

In addition to having OSX Operating System, the Mac OS X Developer Tools must also be installed. These tools are available on the Developer's CD which is included with Operating System software that came with the machine, or it can be downloaded from Apple's Web site. In order to access these tools online, you must become a online member in Apple's Developer Connection, which is free. Consult the web site for the latest version of the Developer tools that are appropriate for the Operating System you are using. As of this writing the latest set of Developer Tools is from December 2002, and they go with OSX 10.2.x. It is important that you have the tools appropriate for your system version, and not necessarily the absolute latest version. The Developer Tools will install among other things a C compiler, which is required to install the DTK.

User Privileges

The user that installs the DTK will need to have administrator rights. This will allow the user to install and test the DTK. Each administrator on an OSX machine is given full 'sudo' privileges so actually becoming root is not necessary.

Locating and using the Mac OS X Terminal Application

Before you install the DTK software you must be at least somewhat familiar with the Terminal application provided with OSX. It is very similar to a shell prompt on a Unix box and is analogous to a DOS prompt on a Windows machine. You can open a Terminal window by clicking on the **Terminal** application which is located in the `/Applications/Utilities` folder. you may wish to resize, or change the features of this window to suit your personal taste. As you install the DTK each of the commands that are listed below will be typed into this Terminal.

Installation

NOTE: While there are several different shells available to the OSX user this document will issue commands to the bash shell. This shell is available on all OSX systems by typing in the word 'bash' at the shell prompt. The default prompt is a '$'. It is beyond the scope of this document to instruct the user on how to use a Unix shell. In addition to the man pages - 'man bash' , one of the best on-line resources available to the user is http://www.gnu.org/manual/bash-2.05a/bashref.html. You can discover the full resources of commands that are available.

Open up the Terminal application by either double-clicking on the app located in the `/Applications/Utilities/Terminal`

Or issue the command

```
$ open /Applications/Utilities/Terminal.app
```

Make Download and Build Directory

```
$ mkdir /dtk-install
$ cd /dtk-install
```

Download the DTK and documentation

You can obtain the DTK from http://www.all.net/dtk. The first step in installing the DTK is to set-up the directories that are needed and then download the DTK itself, which is a TAR file. We will download the DTK into the /dtk-install directory for the installation.

```
$ wget http://all.net/dtk/dtk.tar (or use a web-browser)
$ ls -l
-rw-r--r--  1 imauser  admin  993280 Apr  4 01:38 dtk.tar
```

Once we have it downloaded we need to expand the tar file with the following.

```
$ tar -xvf dtk.tar
@fake.passwd.orig
Configure
download.html
... [~203 other files deleted]
README
```

It is very important not to install the DTK in to the same directory that was downloaded and expanded into. This is noted in the documentation on the Web site and the documentation that comes with the DTK in download.html. For this installation we will install the DTK in the /dtk directory, which is the default.

Building the DTK with the Configure command

Before we attempt to run the Configure command, we must locate the perl lib folder. On most hosts this will be at /usr/lib/perl5, however, on Mac OS X this is not the case. The best way to find this directory is to type in the following command.

```
$ perl -V |grep libpth
libpth=/usr/lib
```

This commanding will tell us exactly where the Perl lib directory can be found. Without the grep statement, this command will list the version of Perl that you're running and how it was built for your platform, as well as the library path and other information.

We are now ready to run the Configure command. It will prompt us for the various parameters it needs to properly install the DTK. The questions from Configure will be in bold type, while the responses that you need are in italics. Be sure to run the command as root, or you will not have the proper permissions to do what you need. You can su to root if you are comfortable with that, but it is not needed for this project. The preferred way to run commands as root is with sudo. You will be asked for your a password when you run the command, enter your password, not the root password. Before each decision that you are asked to

make, I will insert explanatory text in **[THIS TYPE FACE]** explaining my logic for the choices I make during the install.

```
$ sudo Configure
Password:[your USER password – not root]
Copying configure file
Setting paths in files

Install directory (/dtk): /dtk
[lets keep the default as suggested]
Making /dtk...Done.

Which perl (/usr/bin/perl): /usr/bin/perl [this default is
OK]
Which perl lib directory (/usr/lib/perl5):/usr/lib [MAC OS
X only]


Host information.
Which fully-qualified domainname should I claim
(all.net):iamhere.com
[you can use a real one if you have it, but a fake one will
work as well]
Using iamhere.com

Select your real OS from this list (by number) or enter the
OS name (Linux):
  1) Linux        2) Solaris      3) SunOS        4) HPUX
  5) AIX          6) SGI          7) NT           8) Ultrix
  9) SCO
Selection: 1 [OSX is most like Linux on this list – I think
it is a good choice. DTK was programmed a few years before
Mac OS X, so therefore it is not in the list]
OS set to Linux

Select your DECEPTION OS from this list (by number) or
enter the OS name (NT):
  1) Linux        2) Solaris      3) SunOS        4) HPUX
  5) AIX          6) SGI          7) NT           8) Ultrix
  9) SCO
Selection: 2 [This is my preference – I have access to
Solaris – so I can get an idea of how to code realistic
deceptive output]
FAKE_OS set to NT

Log files Standard, sYslog, Compressed, or Database format
(s/y/C/d)?(1): c
```

[A utility for uncompressing these logs comes with the DTK - also they are not compressed like a ZIP file, it is just in a real brief but still intelligible format - I really like that feature]

```
Compressing Logfiles (1)

Password for remote retrieval of the DTK log (all-
characters-no-spaces)
- OR - : for no password)
- OR - O for One Time Pad
- OR - A for Algorithmic Authentication
- OR - T for Time-Based Authentication: (!O)?: qwerty
```

[pick a password]

```
Password set to /qwerty/

Maximum input length to log (250)?: 250
```

[should be enough for SENSIBLE requests]

```
Maximum Length set to 250

Time (in seconds) between inputs before we act like a core
dump (20)?: 20
```

[this is a nice delay - not to long or short]

```
Timeout set to 20

Maximum inputs before we act like a core dump (30)?: 30
```

[this means they get 30 commands at the most - so we can NOT loop them in the state machine forever. This means they CAN NOT tie up system resources to a great extent.]

```
Loop count set to 30

Send email to (root@all.net)?: hereiam@iamhere.com
```

[your email address if you like]

```
Email going to hereiam@iamhere.com
Updating installation

Generate a fake password file based on your password file
(Y/n): y
```

[interesting concept here - this takes the contents of your local /etc/passwd and uses it to generate a fake password file we can send to an attacker - this will not have the expected results on OSX. See below for more information]

```
mv: rename @fake.passwd to @fake.passwd.orig: No such file
or directory
New custom fake password file generated
Done.
```

... **[output lines removed]**

```
Move files in /dtk (Y/n): y
... [output lines removed]
Configure completed - please continue with installation per
the download.html file
```

Modifying the files needed by the DTK

Now we need to modify the files used by the Operating System that decide what services to start, what ports belong to what service, and where to direct traffic on those ports. Thanks to Wietse Vienna's TCP Wrappers, we are able to control - with great precision and ease - what services are allowed to be accessed by whom.

System Preferences / Sharing Pane

Mac OS X has an central application that controls user preferences that is located at `/Application/System Preferences`. One of these panes, 'Sharing', controls what services are available on the machine. You should make sure that a few of these services are enabled, say FTP and WWW, so we can test access to the box and then control it with TCP Wrappers.

The `/dtk-install/download.html` file contains instructions on what files to change to configure a deception service on a particular port. Each of these steps will be examined in turn, and when we need to do something special for OSX, that will be pointed out. The following quotes are taken from the instructions, and the appropriate actions will be listed beneath each one. I will use the command '`sudo pico`' to edit these files. See the man page for `pico` to see what this editor is capable of doing. You can edit these files in your favorite text editor as well, just be sure to save them as plain text.

"*Test TCP wrappers to make sure it works right before you add DTK*. - Cohen"
TCP Wrappers is controlled by the `/etc/hosts.deny` and the `/etc/hosts.allow` files. First we will DENY all services on all ports to all machines, and then selectively permit machines that we trust to access services we provide.

Open the `/etc/hosts.deny` file and add the following line.

```
$ sudo pico /etc/hosts.deny
ALL: ALL
```

Open the `/etc/hosts.allow` file and either make sure the file is blank or place a comment in front of each line. We will test to see if the TCP Wrappers is capable of denying any service, including the ones that have been enabled by the user in the Sharing panel in the System Preferences.

```
$ sudo pico /etc/hosts.allow
#all: all
```

Next we will test if we can access either the FTP or WWW services on the localhost using telnet.

```
$ telnet localhost 21 [trying FTP]
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Connection closed by foreign host.

$ telnet localhost 80 [trying the public Web Server]
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
telnet: Unable to connect to remote host
```

As you can see, these services are not available. If you return to the `/etc/hosts.deny` file and remove the all:all line, you will notice that these services have been restored.

"*Copy the relevant lines from the /dtk/dtk.hosts.allow file into your /etc/hosts.allow file to implement the desired deceptions from addresses not otherwise authorized to perform the applicable services.* - Cohen"

The `dtk.hosts.allow` file comes with several lines that you can include in your `/etc/hosts.allow` file. They are as follows:

```
in.telnetd: all:  twist /dtk/telnetd -L /dtk/Telnet.pl %a 80 %u %d
testing
thttpd:     all:  twist /dtk/Generic.pl %a 80 %u %d testing
in.pop3d:   all:  twist /dtk/Generic.pl %a 110 %u %d unknown
in.wrapd:   all:  twist /dtk/Generic.pl %a 421 %u %d unknown
att111:     all:  twist /dtk/Generic.pl %a 111 %u %d unknown
att10752:   all:  twist /dtk/Generic.pl %a 10752 %u %d unknown
all:         all: twist /dtk/coredump
```

While you can set up deceptive services on as many ports as you like, you need only copy the lines that pertain to you. Of interesting note is that last line, when included as is, indicates that all other services than those listed should be directed to the /dtk/coredump app. This app merely prints a message to the console that a core dump has occurred and then quits. For our purposes to test the DTK operations, we will use most of the lines from this file, and then set up our own unique deception on port 8080. The goal is to leave all existing services in place, get the added protection of deception on the services listed above, and set-up a new deception port of our own.

To do this we will add the following lines to the /etc/hosts.allow file.

```
in.pop3d:   all:  twist /dtk/Generic.pl %a 110 %u %d unknown
in.wrapd:   all:  twist /dtk/Generic.pl %a 421 %u %d unknown
att111:     all:  twist /dtk/Generic.pl %a 111 %u %d unknown
att10752:   all:  twist /dtk/Generic.pl %a 10752 %u %d unknown
```

```
tomcat:     all:  twist /dtk/Generic.pl %a 8080 %u %d unknown
```

I have removed the first 2 lines to keep my existing services in place, which are denying telnet and allowing httpd. We are adding services for ports 110, 111, 421, and 10752, which should not be accessed anyway, and finally adding a port of our own on 8080. Note that I have added the names of the services as are defined in `/etc/services`.

I have chosen port 8080 for a few real-world reasons. I have in the past run an Tomcat server on that port, which is from the Apache Jakarta project. See http://jakarta.apache.org/tomcat/index.html for more information. This is a free, standards-compliant, Java-based web-server, whose default port is 8080. After this service was no longer needed, I was seeing traffic attempting to access 8080, not with HTTP requests for the server, but what appeared to be login attempts. Instead of simply shutting down the port, I decided to run a deceptive service so I could let him in and gather forensic information.

This is what our file looks like now.

```
$ more /etc/hosts.allow
## REAL SERVICES
ftpd:           all
httpd:          all

## DECECPTION SERVICES
in.pop3d:       all:    twist /dtk/Generic.pl %a 110 %u %d unknown
in.wrapd:       all:    twist /dtk/Generic.pl %a 421 %u %d unknown
att111:         all:    twist /dtk/Generic.pl %a 111 %u %d unknown
att10752:       all:    twist /dtk/Generic.pl %a 10752 %u %d unknown

tomcat:         all:    twist /dtk/Generic.pl %a 8080 %u %d unknown
dtk:            all:    twist echo 'The DTK is running here'
```

"*Add the appropriate lines from the /dtk/dtk.inetd.conf file into your /etc/inetd.conf file to enable the services you want to provide deception for.*"

Here we add the line needed to provide a service for tomcat, and direct it to use TCP Wrappers to manage access. This is what the relevant lines in the `/etc/inetd.conf` file look like. The lines that are `/dtk/dtk.inetd.conf` have been copied as well, but have been omitted here for brevity. The twist entry will redirect incoming requests to the services listed, incluing the original parameters passed.

```
$ more /etc/inetd.conf
...
ftp     stream  tcp  nowait  root /usr/libexec/tcpd ftpd -l
dtk     stream  tcp  nowait  root /usr/libexec/tcpd dtk
tomcat stream  tcp  nowait  root /usr/libexec/tcpd tomcat
...
```

We can see that these are being handled by tcpd, which directs requests thru the hosts.allow & hosts.deny files and the name of the service to execute. `ftpd` handles the FTP protocol, while our service will be called `tomcat`. I have also added the `dtk` as a service on its standard port 365, and am returning a message that states it is running. Even if I turn the other deception services off, I can keep this on to keep attackers off balance.

"*Add the appropriate lines from the /dtk/dtk.services file into your /etc/services file to reflect the services you are providing deception for and to add the now official DTK "deception active" port - 365 to your services file."*

The NetInfo Manager

The files that control network operations are managed in a different way on Mac OS X than on any other Unix platform. There is a program called NetInfo Manager, located at "`/Application/Utilities/NetInfo Manager`" which holds the data needed to manage network operations on OSX. This program has the capability of importing and exporting information from the standard Unix system files, so you can in effect use those files in much the same way. Apple has provided command line utilities to accomplish this task. See the man page for nidump and niload for more information beyond what we will use here.

```
$ man nidump
$ man niload
```

This has an interesting side effect. If an attacker does compromise your /etc/passwd file, and you are not keeping that file current with changes in NetInfo Manager, any attacks that are attempted against that file, or any managed by NetInfo Manager, are completely pointless. This, by itself, is a marvelous deception.

We need to use the nidump command to get information out of NetInfo Manager that is normally handled by `/etc/services` and use niload put it back in.

```
$ nidump services . > /dtk-inst/ni_services.txt
```

Now we add the following lines to the file, order is not important.

```
tomcat      8080/tcp
tomcat      8080/udp
```

By looking carefully at the file, or by using grep on the command line, we can see that the deception port, 365, is already registered to the DTK. No further action is needed on our part.

```
$ grep 365 /dtk-inst /ni_services.txt
dtk          365/tcp    #Deception Tool Kit
dtk          365/udp    #Deception Tool Kit
```

Now we put the information back.
`$ niload /dtk-inst/ni_services.txt . < services`

"NOW - find the process of the inet daemon - "ps -a | grep inetd" works on a lot of Unix systems - and send a hangup signal to it to get it to reload the /etc/inetd.conf file into it's memory - "kill -HUP " where is the process Id discovered by the precious ps command does this on many Unix systems."

What this means is get the pid (or Process ID) of the inetd daemon, and then cause it to reload the changes we made to the `/etc/inetd.conf`. On OSX this is accomplished by the following single command.

```
$ kill -HUP `cat /var/run/inetd.pid`
```

This has reloaded the inetd.conf file, and our deception service on port 8080 should give us some type of response, although we sill need to configure it to meet our specific needs.

```
$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
configuration file error - please contact the site administrator
Connection closed by foreign host.
```

Configuring the Response

Custom responses are handled in the DTK thru a `xxx.response` file, where the `xxx` refers to the port number you wish to provide deception on. This must match the number in the `/etc/hosts.allow` file for your service that we configured earlier. We will make our own 8080.response file, and see how that is used to fake a login onto the machine.

The `/etc/hosts.allow` line reads as follows:

```
tomcat:         all:    twist /dtk/Generic.pl %a 8080 %u %d unknown
```

This redirects the request to the Generic.pl script located at /dtk, and the parameters shown are passed including the port number 8080.

State machine

The response file is a finite state machine. The National Institute of Standards and Technology says that "Computation begins in the start state with an input string. It changes to new states depending on the transition function"[REF] . If the input is acceptable or anticipated by matching a string or expression the user is directed to the next state, otherwise an error message or some other action can be taken. The state machines that is used by the DTK is simple to manage and the author should be applauded for his implementation. You can find an example state machine at http://www.all.net/dtk/example.html, which fakes a sendmail daemon.

However don't confuse a simple state machine with knowing a protocol and how-to respond to the different inputs a user attempts. If you are faking sendmail, then having an idea of how sendmail works it in a particular version will increase the value of your deception. In our example we are faking port 8080. There is no protocol for port 8080 that is well known other than for Tomcat which is a web server. The protocol we will fake is simple, and encourages the attacker to go forward, thinking he has succeded. Even if the attacker only gets past the first one or two steps, that will be enough to record IP addresses and possible what the attacker was after.

Let's make a simple response file. You can copy the text below or type it in yourself. These first two lines are comments, but do help with understanding the format of the file.

```
$ sudo pico /dtk/8080.response


#State  Input  State  Exit  lf/file   output/filename
#-------------------------------------------------------------------
```

A line in the response file consists of the following:

```
State       State number - start with 0 and work up
```

```
Input       matches input from the attacker – except these keywords

START – The first response to anythinh
ERROR – what to do if there is no match
NIL – What to do if there is no input
NOTICE –
! -


Next        The next stat to go to. Need not be sequential.

Exit        1 is continue, 0 is exit.

lf/file     0 – no carriage return
            1 – add carriage return
            2 – take response from file specified

output/filename - What the attacker should see.
```

Now we can configure our own file to fake a login. Of course, the more responses you can configure and the more realistic it seems, the higher the value of the deception. Here is the contents of my /dtk/8080.response file with [comments in green].

```
$ more /dtk/8080.response

# State     Input NextState   Exit  lf/file     output/filename
------------------------------------------------------------------------
!     timeout     30    # 30 second timeout per command
!     maxloops    20    # 20 commands at most
#[these are standard]

0     START 0     1     1     Apache Tomcat/4.1.12 - Welcome
#[my welcome message]

0     ERROR 0     1     1     Command unrecognized - use help
#[let them know help is available]

0     help  0     1     1     login help quit bye
#[list a few commands if they ask for help]

0     login 1     1     0     choose a username [root|anonymous]:
#[I saw lines that had 'login' - lets oblidge them]

0     quit  0     0     1     QUIT - CLOSING CONNECTION
0     bye   0     0     1     GOODBYE - CLOSING CONNECTION
0     NIL   0     1     0     ERROR - CLOSING CONNECTION
#[these are my standard ones]

1     root  2     1     0     root @ localhost [ / ] $
1     anonymous   2     1     0     anonymous @ localhost [ / ] $
#[if they typed in these lets give them a psuedo-prompt]

1     help  1     1     1     - login help quit bye
1     quit  1     0     1     QUIT - CLOSING CONNECTION
```

```
1     bye   1     0     1     GOODBYE - CLOSING CONNECTION
1     ERROR 0     0     1     ERROR - CLOSING CONNECTION
1     NIL   2     1     0
#[more common ones]

2     !     2     1     cat   ls    @ls_root.txt
#[the ! matches anything containing the 'ls']
#[@ls root.txt is a text file containing a directory listing]
#[this file has the prompt at the end – continuing the deception]

2     /passwd/    3     1     cat   @ni_passwd.txt
#[the // matches anything containing 'passwd' ]
#[cat is a Unix command that can send contents to the screen]
#[this file has the prompt at the end – continuing the deception]

2     /uname/     2     1     cat   @uname.txt
#[a small clue as to the OS they have accessed]

2     help  2     1     1     login help quit bye
2     quit  2     0     1     QUIT - CLOSING CONNECTION
2     bye   2     0     1     GOODBYE - CLOSING CONNECTION
2     ERROR 0     0     1     ERROR - CLOSING CONNECTION
2     NIL   2     1     0
#[more common ones]

3     NOTICE      notify.pl   Email me@comp.com passwd file accessed
#[the // matches anything containing 'passwd' ]
3     quit  3     0     1     QUIT - CLOSING CONNECTION
3     bye   3     0     1     GOODBYE - CLOSING CONNECTION
3     ERROR 0     0     1     ERROR - CLOSING CONNECTION
3     NIL   3     1     0
```

Whew! That is the end of our response file. As you can see, each request is handled and directed to a state. You can use regular expression matching, and redirect output to other programs. The NOTICE keyword in the last state sends an email if anybody gets around to looking at the /etc/passwd file.

We can test this by using telnet to attempt a login. Another good choice would be to use netcat written by hobbit@atstake. “Netcat has been dubbed the network swiss army knife. It is a simple Unix utility which reads and writes data across network connections…”(hobbit@stake) This type of tool is perfect for these explorations, and does a whole lot more. I will use telnet though, since it is available on all Mac OS X Systems.

```
$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Apache Tomcat/4.1.12 – Welcome [our welcome screen]
help
login help quit bye [the 'help' text]
login
choose a username [root|anonymous]: root [we choose to be root]
root @ localhost [ / ] $ uname [what OS is this?]
```

```
Solaris 2.7
root @ localhost [ / ] $ ls -l [lets list some files]
total 637985
-rw-r--r--   1 root    wheel    488 Apr 02 11:08 passwd -> /etc/passwd
drwxr-xr-x  36 root    wheel   1224 Mar 21 11:54 bin
-rw-r--r--   1 root    wheel   3772 Mar 24 22:14 hosts -> /etc/hosts
dr-xr-xr-x   2 root    wheel    512 Mar 21 18:33 dev
drwxr-xr-x  96 root    wheel   3264 Apr 01 18:34 etc
drwxrwxr-x   3 root    admin    102 Feb  6 07:59 opt
drwxr-xr-x   6 root    wheel    204 Mar 21 18:33 private
drwxr-xr-x  60 root    wheel   2040 Mar  4 10:59 sbin
drwxrwxrwt  11 root    wheel    374 Apr 03 22:32 tmp
drwxr-xr-x  11 root    wheel    374 Mar 12 17:23 usr
drwxr-xr-x  20 root    wheel    680 Mar 21 18:33 var
root @ financehost [ / ] $ cat /etc/passwd [hmmm - this looks easy]
nobody:*:-2:-2::0:0:Unprivileged User:/dev/null:/dev/null
root:ou4LbYGLtFTjI:0:0::0:0:System Administrator:/usr/root:/bin/tcsh
daemon:*:1:1::0:0:System Services:/usr/root:/dev/null
unknown:*:99:99::0:0:Unknown User:/dev/null:/dev/null
www:*:70:70::0:0:World Wide Web Server:/etc/httpd:/dev/null
miked:.0x36LX821Ma2:502:20::0:0:Mike Delano:/usr/miked:/bin/tcsh
samf:.245dfbFGGfgh:502:20::0:0:Samatha Folsome:/usr/samf:/bin/tcsh
jorgem:.dv3b245fDvc3:502:20::0:0:Jorge Matore:/usr/jorgem:/bin/tcsh
mysql:*:74:74::0:0:MySQL User:/dev/null:/dev/null
sshd:*:75:75::0:0:sshd Privilege separation:/var/empty:/dev/null
smmsp:*:25:25::0:0:Sendmail User:/private/etc/mail:/dev/null
newuser:::::0:0:::
root @ financehost [ / ] $ exit [I'm outta here]
ERROR - CLOSING CONNECTION
Connection closed by foreign host.
$
```

This is what an attacker would see if they managed to get to port 8080 on our system and accessed the `/etc/passwd` file.

The DTK log File

Each command was recorded in the log file, along with IP addresses and so forth. Here is what the log file looks like, located at /dtk/log

```
$ more /dtk/log
127.0.0.1 -1 0.0.0.0 8080 2003/04/05 17:53:53 815 815:1 Generic.pl S0
R-0 unknown tomcat unknown
[first two lines are one line - IP addresses, times, ports]
- - - +13 - - 815:1 - S0 R0-0 help
- - - +9 - - 815:1 - S0 R0-0 login
- - - +3 - - 815:1 - S1 R0-0 root
- - - +2 - - 815:1 - S2 R0-0 ls -l
- - - +16 - - 815:1 - S2 R0-0 cat /etc/passwd
- - - +0 - - 815:1 - S3 R@ni_passwd.txt-0 NOTICE notify.pl -1 3 Email
      me@comp.com passwd file accessed
[they got to the /etc/passwd file - if we were running sendmail it
would have sent a message to the address from setup]
- - - +3 - - 815:1 - S3 R1-0 exit
- - - +0 - - 815:1 - S0 R0-0 WeClose
```

After the attack, the DTK exits gracefully and closes the connection.

Summary

We have seen how to install the DTK on Mac OS X, a new operating system by Apple Computer, using the native tools that are distributed with the OS. We have also set-up a deception on a port 8080, which was once used on my machine, and was the target of an attack. The deception was set-up so once they were able to deduce the few commands necessary to access the system, anybody accessing that port would most likely feel welcome. Once they were in, they were treated to a 'root' shell, and access to a few files that should be safely hidden away. Little did they know that each and every move was being carefully recorded, while an alert went sent out when the accessed certain files.

I have shown very experienced Unix System Administrators what I have done on port 8080. Even my simple deception fooled them into a thinking they had obtained a shell prompt with root privileges without entering a password. A few of them were ready to access password files and host lists to discover more about the machine they were on and what vulnerabilities they could exploit. Like most attackers, once they accessed the machine in what they thought was a legitimate shell, they were confident that they could move about freely. Had this of been a real attacker we would have gleaned an enormous amount of information about who they were and what they wanted. That is the initial benefit. Once they realize they are on a deception port they would have to think twice about accessing another port easily on the same box. You can see how setting up many ports on many boxes across the network can easily steal the attackers time and waste effort. They may hesitate mounting further attacks since they will never know if what they're attacking is a really really good deception or the real thing. This is the true value and genius behind the deception toolkit.

References

ManTrap
http://enterprisesecurity.symantec.com/content/displaypdf.cfm?pdfid=292&EID=0

Apple Computer. The Book of VMware: The Complete Guide to VMware Workstation. San Francisco: No Starch Press, 2001.

Spitzner, Lance. Honeypots: Tracking Hackers. Boston, MA: Addison Wesley Professional, 2002

Klug, David. "Honeypots and Intrusion Detection". September 13, 2000
URL: http://www.sans.org/rr/intrusion/honeypots.php (April 2, 2003)

Cohen, Fred. "Deception ToolKit". circa 2001
URL: http://www.all.net/dtk/dtk.html (March 13, 2003)

Cohen, Fred. "Deception ToolKit FAQ". circa 2001
URL: http://all.net/dtk/faq.html (March 13, 2003)

Forristal, Jeff. "Luring Killer Bees With Honey". August 21, 2000
URL: http://www.all.net/dtk/dtk.html (March 13, 2003)

CIAC. "Unix Network Security Tools". circa 2002
URL: http://ciac.llnl.gov/ciac/ToolsUnixNetSec.html#Tcpwrappers (April 3, 2003)

Free Software Foundation."Bash Reference Manual - Table of Contents". *May 03, 2002*
URL: http://ciac.llnl.gov/ciac/ToolsUnixNetSec.html#Tcpwrappers (April 3, 2003)

Black, Paul E. "finite state machine". Feb 12, 2002
URL: http://www.nist.gov/dads/HTML/finiteStateMachine.html (April 4, 2003)

hobbit@stake "Network Utilities". Jan, 2003
URL: http://www.atstake.com/research/tools/network_utilities/ (March 13, 2003)