Creating and Managing Distributed Honeynets using Honeywalls (DRAFT – NOT FOR DISTRIBUTION)

David Dittrich
University of Washington

February 14, 2004

Abstract

The concept of the honeywall goes back to discussions at CanSecWest CORE '01, where the idea was born to use a bootable CD-ROM to implement a layer 2 filtering bridge style device. Customization features were developed to help make it easy for someone with little technical knowledge to set up a honeywall. When combined with other honeywalls, a distributed honeynet can be constructed, but as the distributed honeynet grows in size, a single individual or even a small group cannot monitor and manage it manually. The large number of honeypots, which eventually get compromised, create a burden on forensic analysis resources. By automating critical tasks, a large yet manageable distributed honeynet can be created and used effectively. The issues involved, and customization steps required, are discussed, with examples in appendices to help get you started.

1 Introduction to the Honeywall

1.1 The History of the Honeywall

[Candidate for moving to CD-ROM paper]

In 2003, the Honeynet Project began development of a bootable CD-ROM that implements a layer-2 filtering bridge style firewall, based on a combination of a modified iptables rules, Snort[4] and snort-inline. These patches, scripts, and program source can be found in the tools section of the Honeynet Project web site[1]. We named this system a "honeywall," as it is a hybrid firewall/IDS system designed to implement honeynet data control and data capture features.

The concept of the honeywall goes back to discussions at CanSecWest CORE '01[2], where several members of the Honeynet Project and others (e.g., Theo DeRaat) were discussing ways of making honeynets harder to detect and more flexible. "Get rid of NAT and go layer2 bridge mode," was the consensus. Using layer2 bridging was more transparent and "stealth," and the bridging code – we were convinced by Theo DeRaat we could do everything we needed with OpenBSD – with some tweaks, could allow us to delay/throttle/drop/mess up packets in interesting ways to attain data control (i.e., make it look like a network failure or instability, rather than a predictable "cut off at 10 packets" or "swap 0xabcd with 0x0a0a", as later came about with *hogwash*[5], and now *snort_inline*[10].)

Many of the Project members present were already using OpenBSD[3] as bridging firewalls[11] for personal use on networks at home and work, so this firewall implementation was easy to accept.

Then the idea was born to use a bootable CD-ROM. The reasons for this were many.

One was to make it brain-dead easy for someone to set up a honeynet sensor. No more "which packages do I have to install?" No more manual configuration of applications you need to get a normal system up and running. The honeywall CD comes with pre-configured utilities, things that are needed enabled and things that aren't disabled, special tools already added, etc.

Another was to make it slightly harder to attack. By using a read-only file system to hold part of the operating system, attackers would have to use memory based attacks instead of standard operating system command replacement rootkits. (This is by no means a guarantee the system cannot be successfully attacked, so don't rely on it.)

Yet another was to allow ultra rapid deployment of a honeynet sensor, for example to deal with an incident response scenario. As long as you can locate an x86 compatible PC with a ROM that can boot from a CD device, room for one or two NICS (if you are using a dual-port NIC and have one built-in NIC, you can get by with a single PCI or IDE slot), and a hard drive to hold collected data, you can simply put this device in-line between a suspected compromised host and its wall port and you have transparent inline logging capability.

Around this same time, William Salusky created the Biatchux bootable CD-ROM forensic toolkit (now known as FIRE[12]). FIRE integrates a number of forensic tools in a handy ISO image. It (and all other such distributions) suffer at least one major drawback: It always booted to a clean and unconfigured state and had to be configured manually through a character mode dialog each time you used it. There was no way to customize it for an individual site before burning it to CD-ROM.

A hack was suggested by Dave Dittrich to William that implements "holes" in the ISO image, which is then overwritten with the holes being filled in. (You can find a more complete description of this process in section 4.2). Now the ISO had the flexibility to be customized; to come up and request a DHCP lease, to have custom SSH public/private keys and authorization files, to have a custom password on the root account, and to swap the damn caps lock and control keys!) Each time it booted, it came up a predictable way and was usable remotely right from the start.

Over the next year or so, the vision of the "honeywall" was discussed and debated both within the Project and at invited talks.

People at federal government agencies responsible for protecting the nation's critical infrastructure were very interested in this concept and saw its benefit to their tasks at hand. They started asking questions like, "Could you deploy hundreds, or even thousands of these on different networks – say in the banking sector, or the telecommunications sector – and get sector-wide reconnaissance or early-warning capabilities?" We believed the answer was yes, but knew that the simple "one size fits all" honeywall ISO that required hand configuration would not scale. If you had to have a staff of a hundred people running around the country installing and updating these sensors manually, or had to send everyone who was going to set one up to a five day special course to learn all the required configuration steps, it just wouldn't work. The honeywall had to be more scalable, and to scale it had to be customizable and flexible.

Of course some people won't need to enhance the ISO at all, which is why the choice was made to build in a simple user interface. The box can be used stand-alone that way, and we can document how to use it from the stock ISO. But there are many more interesting things that can be done if the sensor network is increased in size and dispersed widely, which require more careful design and richer feature set in order to pull off.

To scale to a thousand honeywalls and configure them (or reconfigure them if you have to respond quickly to something like a newly publicized vulnerability in Windows RPC/DCOM, OpenSSL, or `rdist`) all within a short time window, requires the systems need to not only be easy to set up and modular in design, but to also have facilities for customization for local network settings and centralized remote management.

The person who sets the honeywall box up should only need the skills to download an ISO image file, burn it to a CD-R, unpack a computer from a box, assemble it, configure the BIOS to boot from CD-ROM, insert the CD-R, plug the right wires into the right holes, and press the power button. Not much more than that. (Granted, to do the local customization it will require some mechanism for modifying the distributed ISO, but a central group can manage that and only deal with distribution of the resulting custom ISOs.) From then on, the honeywall starts logging to a central location and is managed from that central location. Now you have a rapidly deployable sensor network that can be used for all kinds of interesting things.

So those are the basic design goals. Produce something that is easy to install and configure, flexible, extensible, documented so that those who know what they are doing can pick up the ball and run.

A question that may come to mind at this point is "why not use a full-blown commercial OS as our base?" Simple. The concept here is to produce something that is more like an appliance than yet another Linux operating system distribution. A hard drive is thrown into the system because the honeywall is going to log *way more* than a RAM disk can hold, not so we can install the OS on it. Its more of a state and historical data repository than an operating system repository.

In the next section, we take a closer look at what distributed honeynets are, what is necessary to implement and manage a large scale distributed honeynet using honeywalls, and look more closely at the

customization features required to support such a distributed network of honeywall sensors.

2 Distributed Honeynets using Honeywalls

2.1 What is a "Distributed Honeynet?"

As the Honeynet Project has shown over the years, it is possible to deploy a specialized local area network that is instrumented to capture data and control (in various ways) what is done on compromised honeypots within this network. These features, termed *data capture* and *data control*, are defined and described in the paper, *Know Your Enemy: Honeynets*[13].

These elements are also defined for Honeynet Alliance members who wish to implement experimental honeynets, in the form of a requirements document. This can be found in the in the Honeynet Alliance section of the Honeynet Project web site in a paper titled, *Honeynet Definitions, Requirements, and Standards*[14].

There is one more category defined in this requirements document that has had little focus on it until now. That topic is *data collection*, and it comes in to play when you step beyond a *stand-alone honeynet* and consider *distributed honeynets*. We will look at each of these types of honeynets in turn.

2.1.1 A Stand-alone Honeynet

A single honeynet collects data for one or more honeypots within the network. The most common use of the traditional honeynet is by individuals or groups who watch a small set of honeypots within an isolated portion of their network. We can call this a *stand-alone honeynet* for the purposes of this paper. This style of honeynet, shown in Figure 1, is what most people are familiar with today, and when you say "honeynet," this is what most people envision.

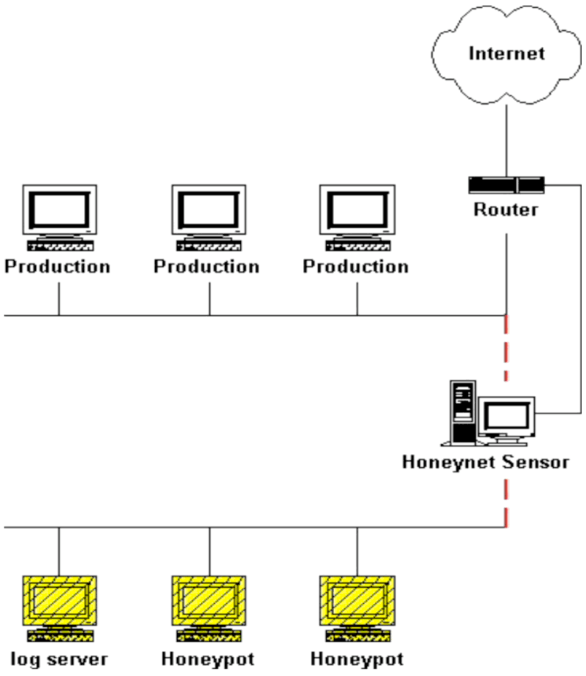

Figure 1: Classic GenII style stand-alone Honeynet

2.1.2 A Distributed Honeynet

A *distributed honeynet* takes this one level higher. Now we are not talking about a single network with one or more honeypots in it, but a group of related honeynets (each with one or more honeypots) that act like a

group of distributed sensors, collecting data in a similar manner to stand-alone honeynets, only consolidating and aggregating some of this data to a central management and logging host for analysis and monitoring at a higher level. This grouping of honeynets into a distributed honeynet can be done within a single organization (e.g., a University or large corporation could deploy a dozen honeynets on a dozen different networks and a single individual or group could manage and monitor them), or a group of Universities or corporations that make up a sector (in the sense of the critical infrastructures as defined by PDD 63[15]) may wish to deploy one hundred honeynets on different networks. This is depicted in the highly simplified diagram in Figure 2. Each of the large clouds shows a separate site, with a single honeywall (the light computer) and honeypot (the black computer.) Note that management connections on the honeywals are not shown, just the bridge going through the honeywall. Site *D* in Figure 2 differs in that it has a grey system, which depicts the central management and logging host for this example distributed honeynet.

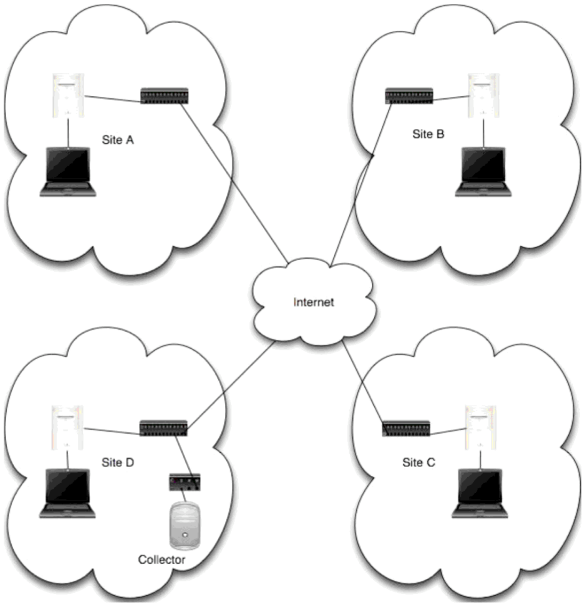

Figure 2: Network of Honeynets at Disparate Sites

In a large scale distributed honeynet built using the Honeynet Project Honeywall bootable CD-ROM, the number of honeywalls may become so large that a single individual, even a group, cannot monitor and manage it manually. Think about having to set up a hundred honeywalls, with a hundred honeypots behind them, then beginning to manage the flood of email alerts they generate on a daily basis, and identifying when one or more of the honeypots (all on different networks, remember) get compromised.

At the point at which the honeypots begin to be compromised, it is necessary to actually analyze them to determine the vulnerability that was exploited, do analysis that produces a report detailed enough to provide "actionable intelligence" to feed back into your defensive operations. The results of the Honeynet Project "Forensic Challenge"[16] showed that an intrusion lasting less than an hour can cause sufficient change to the system that someone would have to spend upwards of 30 hours analyzing the honeypot to determine the full extent of what happened. Multiply this by some percentage of the hundred honeywalls we are discussing here, and consider that two or more of them are compromised in geographically disparate locations, and you can see that analyzing their hard drive contents (as well as identifying and analyzing the network traffic and alerts in the log files collected on the honeywalls) will be daunting process, if you even can muster the resources to do it at all.

The solution *has to be* to automate a large percentage of the work involved in deployment, monitoring, data management, and post-compromise forensic analysis of the honeypots. It has to include remote access to the honeywalls, and it has to include ways to weed through large quantities of data to get the important pieces.

The Honeywall bootable CD-ROM and its customization features are a necessary first step towards

this goal, but they only provide the basic foundation for constructing large scale distributed honeynet deployments.

3 Deployment of honeywalls

We will now consider some of the issues involved in deploying honeywalls. Some of these issues are common to both stand-alone and distributed deployments, so this information may overlap with papers on stand-alone honeynet deployment. Extra attention will be paid here to issues that are specific to distributed deployments to help guide those wishing to pursue this alternative.

3.1 Honeywall Hardware

(Talk about standard hardware and the honeywall hardware guide contents.)

Not finished.

3.2 Managing Honeywall Configurations

The honeywall stores its configuration variables on a local hard drive, so they will last across reboots. It can import the values for these variables using a single text file named (by default) *honeywall.conf*. This provides a mechanism to push a new set of values for a subset of the operational variables (e.g., connection limits, log retention period, etc.) and then cause the honeywall to reconfigure itself based on these new values.

Each ISO can have its own default *honeywall.conf* file, but will all variables be the same for any two deployed honeywalls? Almost certainly the answer is "no."

To simplify distributed management of honeywalls within a single organization, it is helpful to split the variables up into two primary categories: site specific (macro level) variables, and honeywall or honeypot specific (micro level) variables.

Those variables that are site specific can be set the same for all deployed honeywalls, or inherited from a single master configuration file. Examples would be variables for central logging host, limits on connections, snort_inline rules, etc.

Those variables that are honeywall or honeypot specific must be set uniquely for every honeywall. This means that somehow you must keep track of the honeywalls by some unique identifier and produce a configuration file specific to each deployment. Examples would be variables for honeypot IP addresses, management interface IP address, gateway, management DNS servers, honeywall host name, etc.

More work is certainly needed in trying to work out a good way to manage 1000 honeywalls.

[Note: customization of ISOs should include inserting a unique identifier, probably an MD5 hash seeded with the date and other high-entropy data, perhaps in the custom.sh *file. This would then serve as a key for distributed management, correlating honeywall specific configuration settings.]*

Not finished.

A discussion of logging sources and logging issues can be found in Appendix A.

3.3 Deployment of honeypots

The hardware issues for deploying honeypots are much more flexible, and depend greatly on what type of honeypots you wish to deploy. One factor that will lead you to consider hardware standardization for honeypots has to do with cloning of honeypot images for widespread installation (e.g., to attempt to detect "zero-day" exploits across an entire sector or group of institutions.) This will be explained more in section 3.3.3.

One of the time consuming processes is the installation of honyepots to place within your honeynet. The selection of vulnerability profile is one that takes some thought, and unless you are just intending to get your hands on a compromised system to practice cleanup, one that should be considered carefully.

3.3.1 Mirroring production hosts

If your intent is to use a distributed honeynet to monitor for attacks on one of your standard production servers, or your standard desktop installation, the matter is simplified.

Just install the system *exactly the same* as you would the production server or desktop. You want to have the same vulnerability profile, so that an intrusion that is detected by your honeywall will mirror what the attack would (or does) look like on your production systems. You are then in a position to use the full packet capture, connection history, etc. to understand how the attack took place, and can more easily learn the fingerprints of the attack on the production system by analysis of the honeypot (which is not a production system, so will be easier to analyze.)

The analysis of the honeypots can be simplified by use of file system fingerprinting utilities, such as AIDE, Tripwire, L5, etc.[23].

3.3.2 Specific vulnerability profiles

Another specific reason for placing a honeypot on your network is to serve as the proverbial "canary in a coal mine" for zero day exploits.

To serve this purpose, you want the host to be hardened against all known vulnerabilities, but to purposely include one (or more) services that have been discussed publicly as having a known vulnerability.

For example, on November 30, 2003, CERT released an advisory *CA-2003-28 Buffer Overflow in Windows Workstation Service*[24]. This advisory states that proof-of-concept code is available, as well as an exploits, and lists the following systems as being affected

- Microsoft Windows 2000 Service Pack 2, Service Pack 3, Service Pack 4
- Microsoft Windows XP
- Microsoft Windows XP Service Pack 1
- Microsoft Windows XP 64-Bit Edition

Installing any/all of these, or whichever one is most prevalent in use on your network, allows you to then have an early-warning detector of successful attacks against this vulnerability.

Even – or perhaps especially – if there is no known exploit in the wild, yet a vulnerability has been publicly discussed, you may want to deploy a honeypot with this vulnerability. In January of 2002, such an unknown exploit was found in the wild and documented in CERT Advisory CA-2002-01[21]. This exploit targetted Solaris systems running CDE. It was found by a Honeynet Alliance member who detected a compromise to a Solaris honeypot. Network traffic captures were used to confirm the exploit and to create Snort IDS signatures.

3.3.3 Automating honeypot installation

Research is currently being done by a group of graduate software engineering students at Seattle University. This group is implementing a database (named "Manuka," which is the Hawaiian word for honey) to house clean and compromised system images.

The Manuka database is accessed via front-end client applications that are added to a customized bootable CD-ROM that also includes host forensic analysis tools. (Manuka uses both command line and graphic front ends.) The Manuka tools understand how to parse partition tables so that a drive can be effectively mirrored from a honeypot, then later re-installed on a system with a drive the same size or larger. The database also includes attributes of the system that allow searching on operating system type, version, services installed, their versions, and other information that describes the vulnerability profile.

Using the Manuka tools, it only takes one person to install a honeypot with a known vulnerability profile, upload it to the database, and to communicate to others in the organization which image to deploy. Any number of others can then begin automatic installation of this same honeypot image on other systems. This allows for very rapid and consistent honeypot deployment, even in a globally dispersed network.

Constraints on such automated installation include having a hard drive at least as large (or larger) than that on the initial system used to produce the reference image. The operating system being used must also have a method of auto-configuring itself based on hardware found on the system, otherwise you must ensure that the hardware is sufficiently similar to the original system in order for the newly installed operating system to boot. Another consideration is the host IP address, netmask, gateway address, etc. Use of DHCP can make this issue of operating system startup less important, but the existing honeywall implementation requires static IP addresses for honeypots and cannot accomodate use of DHCP at this time.

Not finished.

3.4 Remotely Managing Honeywalls without the User Interface

In order to do automated remote management of honeywalls without using the user interface menu, you must first:

1. Have an established trust relationship for remote command execution,
2. Know which variables must be changed and how to accomplish this from the command line or from within scripts, and
3. Know how to make the changes take effect.

Let's consider each of these items individually.

3.4.1 Remote command execution

In Appendix B, it mentions inclusion of SSH keys and known_hosts file using Type 2 customization. This allows for passwordless remote command execution using OpenSSH. If you have also configured the system to allow remote access to port 22/tcp, you can now execute commands from a central management host.

Not finished.

3.4.2 Changing honeywall variables

Honeywall variables are kept in the directory */hw/conf*, in regular files. This directory functions in an analogous way to the */proc* filesystem in the sense that you can see or change the value of these variables using *cat* or *echo* and I/O redirection. A typical directory would look like this:

```
root@roo2:/dlg# ls /hw/conf
HwALERT                    HwLAN_BCAST_ADDRESS       HwSAVE_ETC_FLOPPY
HwALERT_EMAIL              HwLAN_IFACE               HwSCALE
HwALIAS_MASK               HwLAN_IP_RANGE            HwSEBEK
HwALLOWED_TCP_IN           HwMANAGER                 HwSEBEK_DST_IP
HwALLOWED_TCP_OUT          HwMANAGE_DNS              HwSEBEK_DST_PORT
HwALLOWED_UDP_OUT          HwMANAGE_GATEWAY          HwSEBEK_FATE
HwDISK_INIT                HwMANAGE_IFACE            HwSEBEK_LOG
HwDNS_HOST                 HwMANAGE_IP               HwSSHD_PORT
HwDNS_SVRS                 HwMANAGE_NETMASK
HwSSHD_REMOTE_ROOT_LOGIN
HwHONEYWALL_RUN            HwMANAGE_STARTUP          HwSSHD_STARTUP
HwHOSTNAME                 HwMODE                    HwSWAP
HwHPOT_IP                  HwOTHERRATE               HwTCPRATE
HwICMPRATE                 HwPUBLIC_IP               HwTIME_SVR
HwINET_IFACE               HwQUEUE                   HwUDPRATE
HwINIT_SETUP               HwRESTRICT
```

To see the value of the hostname, for example, you can do this:

```
# cat /hw/conf/HwHOSTNAME
roo2
```

To change the connection limit rate on outbound TCP connections, for example, you can edit the file with your normal editor, or even more simply just redirect output from a shell command, e.g.:

```
# echo 45 > /hw/conf/HwTCPRATE
```

3.4.3 Make changes take effect

After changing variables, it is necessary to restart or reinitialize the service that uses those variables. In the case of the TCP rate limiting variable shown above, that script is */etc/init.d/rc.firewall*. You must do the following to make the change take effect:

```
# /etc/init.d/rc.firewall >/dev/null
```

4 Customization

[Candidate for moving to CD-ROM paper]

So how does the honeywall ISO gain the flexibility needed to implement large scale distributed honeynets? By having hooks for customization that allow things to be added if necessary (e.g., to support a particular custom use of a honeywall) and for the system to be pre-configured to boot a certain way, with particular unique default values (other than the ones supplied by the Honeynet Project.)

There are two primary types of customization of the Honeywall ISO that a site would want to consider. These are called *run-time file system* (or *Type 1*) customization, and *boot-time configuration* (or *Type 2*) customization. As their names imply, Type 1 customization affects the file system contents that will be seen by the system when it is in a running state, and Type 2 customization affects configuration options, variable settings, and other things that control the boot process. (In reality it is a little more complicated than that, but this explanation will suffice for now.)

4.1 Run-time filesystem (Type 1) customization

The first kind of customization of the ISO is the addition, removal, or replacement of programs and files in the *original ISO* to produce an *enhanced ISO*.

You would want to consider Type 1 customization if you wanted to add a set of new programs (and hook into the User Interface to control them, by modifying the existing UI scripts to include options to run your own programs.)

This kind of customization is accomplished using some scripts and `make` files that are bundled in the Honeywall Customization kit.

If you don't care about the details of this kind of customization, you can skip forward to section 4.2 for Type 2 customization, or to section 5 to read about security considerations.

A development host running Linux is required to customize the ISO. We'll call this system the *dev host*. In the directory where you will be customizing the ISO, staging directories are used for incoming files from the *original ISO*, and outgoing files to be made into a new *enhanced ISO*.

Type 1 customization involves ripping apart and reconstructs the ISO image by doing the following:

1. The ISO is mounted into the file system of the *dev host* using a loopback device mount.

   ```
   # mount -o ro,loop -t iso9660 honeywall-0.65.iso /s
   ```

   The contents are now visible to the file system at mount point `/s`:

```
# ls /s /s/fs /s/pkgs
/s:
docs  fs  HoneyWall-v0.65-beta-kanga.ver  isolinux  pkgs

/s/fs:
custom2.sh   lib.cramfs    post-firewall.sh  pre-firewall.sh
custom.sh    post-boot.sh  pre-boot.sh

/s/pkgs:
base.tgz
```

In `/s`, you can see the top level directories that will be mounted as `/mnt/cdrom` in the running honeywall file system.

In `/s/fs`, you can see the cramfs (compressed RAM file system) file for the contents of `/lib` and the "holes" in this ISO. (You can add more, if you want, although you will need to modify `custom.sh` or some other script to invoke them for you.)

In `/s/pkgs`, you can see the compressed `tar` archive that holds everything in the `/etc`, `/usr`, `/sbin`, and `/dlg` directories (which will exist in read/write RAM disk space.)

2. The `lib.cramfs` file system is then mounted on the *dev host* as well, also using a loopback device mount. The uncompressed file files are now accessible to the file system in read-only form, so they must be copied into an incoming staging area on the *dev host*.

3. The `base.tgz` archive is also extracted into the incoming staging area. At this point, the incoming staging area on the *dev host* contains everything from what will be the run-time file system on the honeywall, *except* the initial ramdisk (`initrd`) contents (e.g., things like the `/bin` directory.) Files in the `initrd` file can be over-written in the ramdisk once the system is booted, but can't be changed at this point.

4. You now are free to delete, replace, or add new programs and files into the incoming staging area in the location you want.

5. The compressed RAM file system for `/lib` is compressed into the outgoing staging area, and then deleted. The rest of the files that were previously in the compressed tar archive are now archived again, also located in an outgoing staging area. Finally, `mkisofs` is used to generate a new bootable *enhanced ISO* image.

(Appendix B shows an example of how to do all of these steps.)
This *enhanced ISO* can now be customized for boot-time using the method described in the next section.

4.2 Type 2 customization details

The Type 2 customization features are based on work done by David Dittrich and William Salusky for customizing the FIRE forensic CD. In order to have the CD boot up with custom startup configuration values, a hack was developed by Dave Dittrich wherein the ISO is populated with some Bourne shell scripts that are nothing but a few thousand bytes (to over a megabyte) of comments. These are known as "*holes*" in the ISO. This same method of using holes is employed by the Honeywall ISO.

If you were to look at one of these holes (the file, that is) in an un-customized ISO, it would look like this

```
#!/bin/bash
#<-- custom.sh -->
#------------------------------------------------------------------------------;
#------------------------------------------------------------------------------;
#------------------------------------------------------------------------------;
```

```
#--------------------------------------------------------------------------------;
#--------------------------------------------------------------------------------;
              [many lines removed]
#--------------------------------------------------------------------------------;
```

The hole, in this case, is identified by the name `custom.sh` between the `<--` and `-->`. This hole will be filled by the file (if it exists in the customization directory) with that name.

A simple Perl script locates these "holes" in the ISO, and fills them in with commands of the user's choosing (i.e., the contents of files with those same names in the customization directory.)

This provided a way to customize things like switching caps lock and control keys, obtaining a DHCP lease (or defining a static IP address, netmask, host name/IP mappings, etc.), changing the default root password, and loading in a limited set of new programs into writeable areas in the system's RAM disk.

This technique provides a means of customizing the boot-time configuration of the CD, but has very limited ability to add new programs to the system. (Granted, you could keep adding more holes or make them bigger, but that simply uses up space in the ISO that may not be necessary.) This mechanism also suffers from the problem that you can't remove programs from a read-only compressed filesystem. You are only able to load files into RAM disk areas (which is why the previous method of customization is also required.)

An example of using the *pre-boot.sh* customization hole can be found in Appendix B.

We will now examine the security considerations encountered when using the Honeywall CD in a production capacity, whether it is stand-alone or distributed.

5 Security Considerations

Using a bootable CD-ROM based operating system has some physical security issues that aren't *quite* as bad as with normal systems. At least you can protect workstations that normally boot from a hard drive by using BIOS passwords and pre-defined boot device selection to try to secure the hard drive (although someone could just walk off with the whole system and get access to it by opening the case.)

When booting from a CD-ROM as the primary boot device, it is relatively easy to get the drive door to open and eject the disc, which an attacker can then plop into another system and copy and/or analyze. Or someone may throw away an old ISO or have it "walk off" if left lying on a desk.

What is more, *the CD boots directly into the user interface, allowing manipulation of the Honeywall without requiring a password!* Best practice here is to lock the system into a physical enclosure so that nobody can get to it, physically destroy all old ISOs (or better yet, cycle CD-RW discs), and lock up any CDs that are not currently being used.

Beyond the physical security issues, there are some network related issues as well. Some of these can be eliminated by using the honeywall in a one-off deployment (meaning one honeywall and no management host, or a management host on an isolated network, as depicted in Figure 3.)

When using distributed honeywalls, there will be some security issues that present themselves that are not found in stand-alone installations (or in installations where cross-over cables are used to connect directly to the management host.) These have to do with exposure of information in transit (e.g., log data, email alerts) as well as detection of the presense of the honeywall by passive monitoring of traffic and noting contemporaneous traffic being generated when the attacker probes hosts that are actually honeypots behind the honeywall. On a very quiet network, this would stick out like a sore thumb.

5.1 Passwords and SSH keys

The honeywall ISO is delivered with a default password. When you use the initialization interview in the user interface, it will ask you to change root's password. You can avoid needing to do this by using the customization features described in section 4 and Appendix B.

It does not come with any SSH keys (you can generate them using the user interface if you want, but it may be simpler for large installations to have static keys that are automatically included in the ISO.)

5.2 File system security

The most vulnerable information are the secret password hashes, private SSH keys, etc., that you wish to have exist in the run-time file system. Since the CD-R/CD-RW is not encrypted, you must make *sure* that nobody can get their hands on a copy. (This is another reason why there are two directories used in customization, `./root/` for new programs, and `./root.local/` for local files. That is, a site may distribute its scripts for adding new programs to the honeywall ISO, but may wish to keep all SSH keys and shadow files unique to each customized ISO.)

5.3 Network access security

The honeywall's two bridge interfaces do not have IP addresses, so they are not directly accessible on the network. They simply bridge traffic from one interface to another, albeit sending it through iptables, the kernel bridge code, and snort/snort-inline. An attacker could conceivably exploit a vulnerability in one of these and gain remote root access by injecting commands directly within packets without ever establishing a TCP stream. The answer to these issues is to *patch, patch, patch*, which means keeping up with the releases of the honeywall CD-ROM.

The management interface, on the other hand, *does* have an IP address. This means you should protect it as much as possible, both from direct attack and also from passive monitoring of command/control and logging traffic. We'll get to a solution to this in a moment.

5.4 Network traffic security

Even if encryption is used, the fact that logging is occuring can provide an attacker with a feedback loop that could allow them to detect that a honeywall is in place, even though it is not "visible" between the network gateway or router and the honeypots behind it.

The easiest way to secure the traffic coming/going through the management interface on a honeywall is by separating the management and logging traffic from honeypot traffic. This could be accomplished using VLANs on a switch, using a cross-over ethernet cable and connecting directly to a second interface on the management host, or by using a second network. The result is that no traffic related to the honeywall will transit on the same segment as the honyepots, so the emanations problem mentioned above does not occur.

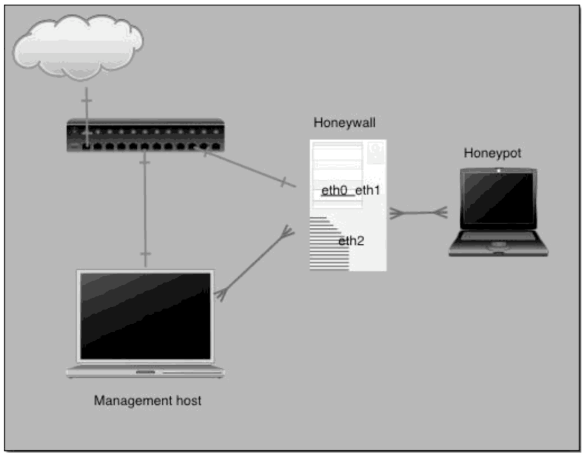

Figure 3: Honeywall with direct management via cross-over cable

Another issue to keep in mind is support protocols, like DNS. If you do things like reverse name lookups to see DNS names in logs, and you do this from the honeywall, you provide an attacker with another way to determine that they have contacted a honeypot, *and* they have a way to determine what IP address is being used by the management interface (which is where the reverse DNS requests will come from.) It is

best to *avoid* reverse DNS lookups from the honeywall, and to use static maps in the */etc/hosts* file as much as possible.

But what if you have to manage the honeywall remotely, and do so on the same physical wire? In this case, there are a few possibilities that should be considered:

1. Using VLAN tunneling to separate the traffic. (This may not be entirely secure from attack, since switches have been known to have exploitable weaknesses that remove the VLAN separation.)

2. Tunnelling all traffic to the central logging/management host(s) using IPSec with ESP. This still puts traffic in-band, so a method of generating random traffic at periods when the honeywall is not sending out log or alert information would make detection by timing more difficult. (There are no existing facilities for doing this.)

5.5 Denial of Service

Another potential problem with a network traffic and IDS logging device is denial of service. A paper by Thomas Ptacek and Tim Newsham[25] on eluding network intrusion detection covers some of these issues.

Basically, if the amount of data being generated in the form of alerts and logs on the honyewall exceeds the I/O capacity of the drive or the CPU or interrupt capacity, the honeywall will either start dropping things, introduce visible latency in bridging traffic, or lock up or crash and stop routing any traffic at all. These all mean an attacker can defeat the logging or detect the presence of the honeywall.

Ways to deal with this situation include any/all of the following:

1. Buy the fastest hard drive possible and avoid building the system in ways that restrict I/O capacity.

2. Tune the hard drive using `hdparm`[26] to increase its I/O capacity. `hdparm` has been included in the honeywall ISO for this purpose.

3. Use slower NICs for the bridging interfaces than the management interface (e.g., 10Mbps for at least one bridge interface and 100Mbps or GigE for the management interface.)

4. Use slower NICs on all honeypots. (This won't decrease the amount of traffic that comes into the honeywall, but it will decrease the amount of traffic overall through the bridging interfaces.)

5. Use an old slow (10Mbps) switch or hub on the external side of the honeywall bridge between the wall port and the external bridge interface.

6. Throttle traffic using bandwidth rate limiting features[27] like Cisco's Committed Access Rate (CAR), Juniper's Traffic Policing, etc. in your routers.

6 Conclusion

So in conclusion, we...

Acknowledgements

Acks

Biographies

Dave Dittrich is an affiliate Information Assurance researcher for the Information School at the University of Washington. He spent over 12 years supporting Unix workstation administrators on the University of Washington campus, leading into the role of Senior Security Engineer. His credits include technical analyses (alone or in teams) of the Trinoo, Tribe Flood Network, Stacheldraht, shaft, and mstream distributed denial

of service (DDoS) attack tools, and many papers on topics such as intrusion response, rootkits and post-intrusion concealment, forensic analysis of hosts and network traffic, and incident cost analysis.

Dave is now heavily involved in setting up an Information Assurance research program at the UW, and doing research and teaching courses for the Information School.

His home page can be found at `http://staff.washington.edu/dittrich` and can be reached at `dittrich [at] u.washington.edu`.

A Logging

[Candidate for moving to CD-ROM paper]

A.1 Sources of log data

The current implementation of the honeywall CD-ROM logs the following information:

1. Snort alerts

   These logs show attacks that are detected by the Open Source Snort[4] intrusion detection system. Snort logs are placed in directories with names composed of the year, month, and day in numeric format (to allow segregation by day, with natural sorting). For example, the logs for January 8, 2004 would show up in the directory `/var/log/snort/20040108`.

   An example of the files created for this day is

```
root@roo-uw1:/var/log/snort/20040108# ls -lat
-rw-------    1 root     root        41474 Jan  8 10:30 snort.log.1073557485
drwxr-xr-x    2 snort    root         4096 Jan  8 10:24 .
-rw-------    1 root     root          210 Jan  8 10:24 snort_inline-fast
-rw-------    1 root     root          583 Jan  8 10:24 snort_inline-full
-rw-------    1 root     root      1054452 Jan  8 10:22 snort.log.1073552300
-rw-------    1 root     root       912229 Jan  8 10:20 snort_fast
-rw-------    1 root     root      1587119 Jan  8 10:20 snort_full
drwxr-xr-x    3 root     root         4096 Jan  8 08:47 ..
```

   The file `snort_full` contains standard Snort IDS entries that look like:

```
[**] [111:10:1] (spp_stream4) STEALTH ACTIVITY (XMAS scan) detection [**]
01/08-10:06:09.729583 10.10.10.3:46271 -> 10.10.10.10:1
TCP TTL:52 TOS:0x0 ID:29436 IpLen:20 DgmLen:60
**U*P**F Seq: 0x452BBA60  Ack: 0x0  Win: 0x400  TcpLen: 40  UrgPtr: 0x0
TCP Options (4) => WS: 10 NOP MSS: 265 TS: 1061109567 0
```

   The equivalent log entry from `snort_fast` would look like:

```
01/08-10:06:09.729583  [**] [111:10:1] (spp_stream4) STEALTH ACTIVITY
(XMAS scan) detection [**] TCP 10.10.10.3:46271 -> 10.10.10.10:1
```

2. snort_inline alerts

   These alerts show up in the same directory as the other Snort logs. An example would look like:

```
03/23-21:21:05.915340 [**] [1:0:0] Dropping Telnet connection [**]
[Priority: 0] {TCP} 10.10.10.10:39528 -> 192.168.1.20:23
03/23-21:21:24.054533 [**] [1:0:0] Modifying HTTP GET command [**]
[Priority: 0] {TCP} 10.10.10.10:38533 -> 192.168.1.20:80
```

3. Full packet capture for all traffic

   The file *snort.log.1073552300* in the directory listing above contains all of the traffic through the honeywall bridge interface during this time period. This file is in PCAP format[6] and can be read with programs such as *tcpdump*[7], *ngrep*[8], etc. (Note that when snort is restarted, it will create a new snort.log.* file. There may be several in each directory as a result. They can be combined back into a single file using *tcpslice*[9].)

4. iptables connection limits alerts

   ```
   Jan  9 10:02:27 honeywall user.warn klogd: Drop TCP after 9 attemptsIN=br0
   OUT=br0 PHYSIN=eth1 PHYSOUT=eth0 SRC=10.10.10.10 DST=10.10.10.2 LEN=60
   TOS=0x00 PREC=0x00 TTL=64 ID=32932 DF PROTO=TCP SPT=32830 DPT=9999
   WINDOW=5840 RES=0x00 SYN URGP=0
   ```

5. iptables firewall rule matches

   iptables rule matches are handled by syslog. The file they are logged to is `/var/log/messages`.

   An example entry looks like:

   ```
   Jan  8 09:52:43 honeywall user.warn klogd: INBOUND ICMP: IN=br0
   OUT=br0 PHYSIN=eth0 PHYSOUT=eth1 SRC=10.10.10.3 DST=10.10.10.10 LEN=84
   TOS=0x00 PREC=0x00 TTL=64
   ```

6. Sebek keystroke logs

   Sebek logs are not be covered here. See [22] for more.

7. Service events using syslog

   Similarly to iptables, services that log by syslog also have their entries placed in `/var/log/messages`.

   An example entry looks like:

   Example here

A.2 Logging and Alerting Triggers

Not finished.

A.3 Logging rates

Not finished.

A.4 Centralized Logging and Alerting

Not finished.

A.5 Attacks on honeywall logging

Not finished.

B Adding New Programs to the Honeywall ISO

Let us look now at an example of how to augment the Honeywall ISO to include new programs of your own chosing. The program (and accompanying menu entry in *Status.sh* to support it) is `tcpdstat`[28].

Here are the steps you should follow:

1. Create the customization directory, and install the most recent release of the honeywall customization tools. In this example, we are using `honeywall-custom-51.tgz`. Create a directory in which to do your customization, like this:

   ```
   # cd /usr/local/src/honeywall/customize
   # wget http://staff.washington.edu/dittrich/misc/honeywall-custom-51.tgz
   # tar -xf /root/honeywall-custom-51.tgz
   ```

2. Copy the latest version of the honeywall ISO

   ```
   # cp ~/honeywall-0.65a.iso .
   ```

3. The customization tools use `make` and some scripts to configure and control everything. You first need to configure it to know about the name of the distributed Honeywall ISO you are going to use (in this case *honeywall-0.65a.iso*), the name you want to use for your enhanced ISO (let's call it *hw-0.65a.iso*), the directories in which you will place files for type 1 (*./root*) and type 2 (*./root.local*) customization files, the name of the Honeywall configuration file you wish to use (default is *honeywall.conf*, and the IP address you will use for your Honeywall if you are doing live development testing.

   To configure, do the following, answering the questions it asks:

   ```
   # make configure
   ######################################################################
   You are about to (re)configure scripts used for enhancing
   a stock Honeywall ISO and/or for customizing your own site-
   specific honeywall ISO images.

   Use Ctrl-C now if you do not wish to do this.
   ######################################################################

   Which ISO would you like to customize? [honeywall-0.65.iso]:
   What do you want to call the enhanced ISO? [hw-0.65.iso]:
   Which directory holds programs to add to the ISO? [./root]:
   Which directory holds files for customizing the ISO? [./root.local]:
   Which import file would you like to use? [honeywall.conf]:
   What IP address is your development test honeywall? [10.10.10.50]:

   Configuring Makefile
   Configuring Makefile.iso
   Configuring isoutil
   ```

   In this example, we had already entered the name `honeywall-0.65.iso` as the *original ISO*, `hw-0.65.iso` for the name of our *enhanced ISO*, etc.

   Once you have set these values, they are saved and will serve as defaults in future.

4. To get access to files on the Honeywall ISO that you wish to modify (in this case, */dlg/Status.sh*), use the command:

```
# make unpack
```

You now have a file tree that contains all of the files (minus things in the initial ram disk) in the directory `./iso-in-stage/root`.

5. Copy all of the files you want to add to this honeywall ISO image into a sub-directory named `./root/` in the customization directory. This directory is something like that used in in a `chroot` environment, where the root of the run-time file system is just a subdirectory when viewed from "outside."

   The `find` command just shows the paths to files in `./root/` for illustration purposes.

```
# mkdir root
# (cd /usr/local; tar -cf - .) |
> (cd root; tar -xf -)
# find ./root -type f
 . . .
./root/usr/local/bin/tcpdstat
./root/usr/local/sbin/ntop
 . . .
```

6. Copy the script that implements the new feature you want, in this case to identify all Snort log files and provide a menu interface to select which one to obtain statistics on using *tcpdstat*. (A copy of a script to do this can be found at `http://staff.washington.edu/dittrich/misc/tcpdstat.sh`) Let's just use this file, shall we? And just to make sure we have a directory in which to put the file, we will use *mkdir* to create it first.

```
# mkdir -p root/dlg
# wget http://staff.washington.edu/dittrich/misc/tcpdstat.sh
# mv tcpdstat.sh root/dlg/
```

7. Now copy the *Status.sh* file from the staging area into the directory where your type 1 customization files will be kept (that is, into `./root/dlg` in this case.)

```
# cp iso-in-stage/root/dlg/Status.sh root/dlg/Status.sh
```

   You are now ready to edit the file. (You can find a copy of *Status.sh* already modified to run our new */dlg/tcpdstat.sh* script, at `http://staff.washington.edu/dittrich/misc/Status.sh`)

8. To add the files in `./root/` to the ISO, you use the following command:

```
# make enhanced
```

   It will inform you of what it is doing along the way. If it succeeds, you will see:

```
# make enhanced
 . . .
Max brk space used a000
21040 extents written (41 Mb)
-rw-r--r--    1 root     root     43089920 Dec 13 18:51 hw-0.65-beta.iso
30669e354f0d617faf6249b939599e8e  hw-0.65-beta.iso
make[1]: Leaving directory ‘/w/honeywall/customize’
```

9. From this point on, as long as you aren't trying to add new programs to the original ISO, you can just use the type 2 customization features to alter the boot configuration (which also allows a limited addition of files, provided they total less than 1MB.)

   Some things you will want to consider for type 2 customization (shown with the path relative to the customization directory, so you know where to put them) include:

   - Custom Network Interface Card (NIC) module installation list in `./root.local/etc/nicmodlist` (This is useful if you have NICs from different manufacturers and want to force device designations, e.g., on a Shuttle development system with a built in 10/100Mbps NIC that uses the *8139too* driver, and an Intel S100 Dual Port server NIC that uses the *eepro100* driver, using a `nicmodlist` that contains "`eepro100 8139too`" forces the Intel card to get *eth0* and *eth1* for the bridge, and the built-in NIC to get *eth2* for the management interface. Of course you can control which interfaces are used for which purpose, but it is convenient to have the bridge clearly marked *IN* and *OUT* on the A and B ports of the Intel dual NIC.)
   - Custom passwords in `./root.local/etc/shadow` (Create this file by making a bogus account in `/etc/passwd` and changing its password, then editing the line to have "root" as the account name. It is advised to set your own default password to avoid someone using the default to access the honeywall before you change it manually.)
   - Custom SSH keys in `./root.local/home/root/.ssh` (This simplifies remote administration by allowing remote command execution and tunnelling using SSH.)
   - Known SSH hosts keys and ssh client/server configuration files in `./root.local/etc/ssh`
   - Customized swatch rules file in `./root.local/etc/swatchrc`

   Just like you placed files in a directory hierarchy named `./root/` to add new programs to the enhanced ISO, you create a directory called `./root.local/` for adding custom configuration files such as those listed above. This way, your *enhanced ISO* does not include these sensitive files, and can be given to someone else within your organization to customize on their own. (See section 5 for security considerations regarding customized honeywall ISOs and CD-ROMs.)

   Here is what a typical directory hierarchy might look like:

```
# find root.local -type f
root.local/etc/ssh/sshd_config
root.local/etc/ssh/ssh_config
root.local/etc/ssh/ssh_host_rsa_key
root.local/etc/ssh/ssh_host_rsa_key.pub
root.local/etc/ssh/ssh_host_dsa_key
root.local/etc/ssh/ssh_host_dsa_key.pub
root.local/etc/nicmodlist
root.local/etc/shadow
root.local/etc/swatchrc
root.local/home/root/.ssh/id_dsa
root.local/home/root/.ssh/id_dsa.pub
root.local/home/root/.ssh/id_rsa
root.local/home/root/.ssh/id_rsa.pub
root.local/home/root/.ssh/fingerprints.txt
root.local/home/root/.ssh/identification
root.local/home/root/.ssh/authorized_keys
root.local/home/root/.ssh/known_hosts
```

10. Other customizations to control boot-time behavior can be done using one of the following "holes" (empty shell scripts, which are executed in this order at boot time): `custom.sh`, `pre-boot.sh`, `pre-firewall.sh`, `post-boot.sh`, `post-firewall.sh`

    An example of using `pre-boot.sh` to develop the floppy disk configuration features looked like this:

```
#
# Example pre-boot.sh script for honeywall CD-ROM.
#
# Dave Dittrich <dittrich@speakeasy.net>
# Fri Jan  9 01:11:18 PST 2004

CONFDIR="/hw/conf"

# Load defaults for this Honeywall.  Priority is:
# 1).  From existing /hw/conf directory,
# 2).  From "honeywall.conf" file on floppy disk, or
# 3).  From /etc/honeywall.conf file (loaded from CD-ROM if you place
#      it there via the custom2.sh file -- see Makefile)

# Note: This file will be found by driveinit.sh and used then.
# dittrich 01/08/04
mcopy a:/honeywall.conf /tmp >/dev/null 2>&1
if [ $? -eq 0 ]; then
    echo ""
    echo
"************************************************************"
    echo "         I found a honeywall.conf file on floppy."
    echo "         Leaving it in /tmp for now..."
    echo
"************************************************************"
fi
echo ""
sleep 5
exit 0
```

11. Create a new CD-RW from your *enhanced ISO* using this command:

    ```
    # make cdrw
    ```

    (Substitute `cdrom` for `cdrw` if you wish to write a CD-R instead of a CD-RW. CD-RW is suggested if you are going to be doing this frequently for testing, as it saves on wasted plastic.)

    You now have a customized honeywall CD-RW ready to boot. To test the new feature we added to run *tcpdstat*, see the new final item in the Status menu.

B.1 Example of altering an existing Honeywall script

We will now take a look at an example of using type 1 customization to alter the functionality of the */etc/init.d/snort.sh* script.

```
# make unpack
make[1]: Entering directory ‘/w/honeywall/customize’
honeywall
Cleaning out staging directories...Done.
Copying files from ISO to iso-in-stage directory...lib...Done.
base...Done.
Copying files from ISO to iso-out-stage directory...Done.
Unmounting /s ...Done.
Ready to ’make update’
```

```
(Make sure you've coped files into ./root first, though.)
make[1]: Leaving directory `/w/honeywall/customize'

# find iso-in-stage -name swatch.sh
iso-in-stage/root/etc/init.d/swatch.sh

# mkdir -p root/etc/init.d

# mv iso-in-stage/root/etc/init.d/swatch.sh root/etc/init.d/swatch.sh

# vi root/etc/init.d/swatch.sh
 [make your edits and save.]

# make update pack
make[1]: Entering directory `/w/honeywall/customize'
cp README.custom-iso ./root/usr/docs/customize
honeywall
Adding files from ./root..../
./etc/
./etc/init.d/
./etc/init.d/swatch.sh
. . .
Done.
Ready to 'make pack' to generate hw-0.65.iso
make[1]: Leaving directory `/w/honeywall/customize'
make[1]: Entering directory `/w/honeywall/customize'
honeywall
Generating new lib.cramfs...
Directory data: 96636 bytes
Everything: 27456 kilobytes
Super block: 76 bytes
CRC: 564c3a75
Done.
Generating new base.tgz...
Done.
Creating new ISO image...

Size of boot image is 4 sectors -> No emulation
 23.81% done, estimate finish Fri Jan  9 16:51:49 2004
 47.65% done, estimate finish Fri Jan  9 16:51:49 2004
 71.43% done, estimate finish Fri Jan  9 16:51:49 2004
 95.24% done, estimate finish Fri Jan  9 16:51:49 2004
Total extents actually written = 21006
Total translation table size: 2048
Total rockridge attributes bytes: 2899
Total directory bytes: 8192
Path table size(bytes): 60
Max brk space used 8000
21006 extents written (41 Mb)
-rw-r--r--    1 root     root     43020288 Jan  9 16:51 hw-0.65.iso
2b070e774bcc418ca906e8347b8017c6  hw-0.65.iso
make[1]: Leaving directory `/w/honeywall/customize'
```

References

[1] Honeynet Project Tools page http://project.honeynet.org/tools/index.html

[2] CanSecWest CORE '01 http://www.cansecwest.com/archives.html

[3] OpenBSD Operating System http://www.openbsd.com/

[4] Snort Open Source Intrusion Detection System http://www.snort.org/

[5] Hogwash - H2 Based Packet Scrubber http://hogwash.sf.net/

[6] PCAP library (libpcap) http://www.tcpdump.org/

[7] tcpdump http://www.tcpdump.org/

[8] ngrep (Network grep) http://sourceforge.net/projects/ngrep/

[9] tcpslice ftp://ftp.ee.lbl.gov/tcpslice.tar.gz

[10] snort_inline http://snort-inline.sf.net/

[11] OpenBSD bridge without IPs using IPF Tutorial, by Doug Hogan and Bryan Hinton, DaemonNews http://www.daemonnews.org/200103/ipf_bridge.html

[12] FIRE bootable forensics CD http://fire.dmzs.com

[13] Know Your Enemy: Honeynets http://www.honeynet.org/papers/honeynet/

[14] Honeynet Definitions, Requirements, and Standards http://www.honeynet.org/alliance/requirements.html

[15] http://www.ciao.gov/resource/paper598.pdf

[16] The Forensic Challenge http://www.honeynet.org/challenge/

[17] CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University http://www.cert.org

[18] Bugtraq mailing list at SecurityFocus.com http://www.securityfocus.com/archive/1

[19] full-disclosure mailing list at Netsys.com http://www.netsys.com/full-disclosure/index.html

[20] Incidents mailing list at SecurityFocus.com http://www.securityfocus.com/archive/75

[21] CERT Advisory CA-2002-01 Exploitation of Vulnerability in CDE Subprocess Control Service, January 14, 2002 http://www.cert.org/advisories/CA-2002-01.html

[22] Know Your Enemy: Sebek, The Honeynet Project http://www.honeynet.org/papers/sebek.pdf

[23] Integrity Checker web site http://wwwpdp.web.cern.ch/wwwpdp/as/security/general/tools/integrity.html

[24] CA-2003028 Buffer Overflow in Windows Workstation Service http://www.cert.org/advisories/CA-2003-28.html

[25] Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection, by Thomas Ptacek and Tim Newsham

[26] Tuning IDE Hard Disk Performance http://www.faqs.org/docs/securing/chap6sec74.html

[27] Honeypot Bandwidth Rate Limitation (incomplete draft), 05/13/2002, by Edward Balas http://project.honeynet.org/papers/honeynet/tools/dc.html

[28] tcpdstat (UW modified version) http://staff.washington.edu/dittrich/misc/tools/tcpdstat-uw.tar.gz