The Honeywall from 30,000 feet: Honeywall Fundamentals

George Chamales
University of Texas Austin

David Dittrich
University of Washington

March 21, 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 History

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[1] and snort-inline. These patches, scripts, and program source can be found in the tools section of the Honeynet Project web site[2]. 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[3], 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*[4], and now *snort_inline*[5].)

Many of the Project members present were already using OpenBSD[6] as bridging firewalls[7] 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[8]). 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. 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.

2 Basic Components

Describe Data Control and Data Capture [15].

6.1 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.

6.2 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 6.2.3.

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.

6.2.1 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.[9].

6.2.2 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*[17]. 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[16]. 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.

6.2.3 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.

6.3 Privacy

6.4 Downstream liability

6.4.1 "But you knew it was being used to attack me!"

6.5 Collateral damage to other internal systems

6.6 Plan ahead (Richard's "Collateral Damage" section)

7 Deployment scenarios

7.1 Simple GenII Honeywall

7.2 Forensic analysis of compromised host

7.3 Traffic analysis (no snort packet logging)

A Logging

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[1] 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[10] and can be read with programs such as *tcpdump*[11], *ngrep*[12], 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*[13].)

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 [18] 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.

References

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

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

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

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

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

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

[7] OpenBSD bridge without IPs using IPF Tutorial, by Doug Hogan and Bryan Hinton, DaemonNews

[8] FIRE bootable forensics CD http://fire.dmzs.com http://www.daemonnews.org/200103/ipf_bridge.html

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

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

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

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

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

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

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

[16] 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

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

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