

BSD Honeypots

Of course it runs on BSD



A discussion on the use of BSD Operating Systems to deploy honeypots.

Michael Shirk

BSDCan 2020

@Shirkdog <https://github.com/Shirkdog>

Agenda

- Introduction
- What is a Honeypot?
- History of Honeypots
- Honeypots on BSD
- Case Study of a Honeypot on FreeBSD
- Recent Examples of Honeypots in Security Research
- Conclusion

Rant Warning

- Whenever you see beastie with a hammer, I may be ranting without any evidence.
- All information not cited in this talk is based on personal experience or opinion (marked with an asterisk *).



Introduction

- Worked in IDS/IPS since 2003 (various positions including consulting)
 - Engines: Snort, Suricata, Dragon, Zeek (formerly Bro) (also had to work with McAfee, ISS, NFR ... others)
 - Signatures for Emerging Threats (since they were Bleeding Edge Snort)
 - Reverse Engineering/Exploit Development (Mostly on Windows, which is why I hate it.)
- Support Open Source Security Tools and Software
 - Maintain pulledpork for Snort/Suricata (rule updating script):
 - <http://github.com/shirkdog/pulledpork>
 - Active community member of open source projects:
 - Operating Systems: FreeBSD, OpenBSD, HardenedBSD
 - Security Tools: Snort, Suricata, Zeek, AIDE
 - Maintainer of several FreeBSD ports.

Introduction

- Senior Consultant/Co-Owner of Daemon Security, Inc.
 - Use HardenedBSD, OpenBSD and FreeBSD daily for my work, but customers mostly use Linux.
- Assistant Professor of Cybersecurity at Harford Community College
- Founder of CharmBUG – Baltimore Area BSD User Group
 - Founded in 2016 <http://www.charmbug.org>

What is a Honeypot?

- Basic definition – a pot of honey that attracts things, like bears or bees.
- More useful definition for computers:

“A honeypot is a closely monitored network decoy serving several purposes: it can distract adversaries from more valuable machines on a network, provide early warning about new attack and exploitation trends, or allow in-depth examination of adversaries during and after exploitation of a honeypot.”
(Provos, 2003).

What is a Honeypot?

- Breaking this down:
 - Distract adversaries from more valuable machines
 - Deploy 30 systems, but only 10 are actual systems, the others are honeypots simulating your actual systems as decoys.
 - Early warning for new attacks/exploit
 - A honeypot is deployed and catches a new exploit attempt for an HTTP server vulnerability.
 - Examination of adversaries during and after exploitation
 - An attacker interacts with the honeypot after an exploit attempt and downloads additional tools, providing additional IOCs for security researchers.

What is a Honeypot?

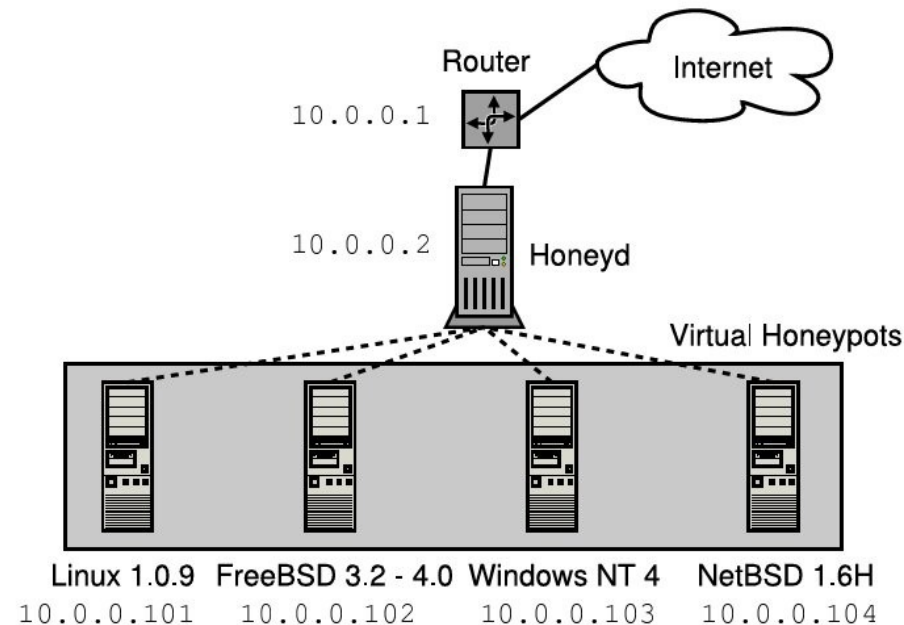


Figure 1: Honeyd receives traffic for its virtual honeypots via a router or Proxy ARP. For each honeypot, Honeyd can simulate the network stack behavior of a different operating system.

(Provos, 2003, p. 3)

What is a Honeypot?

- Honeypots provide benefits for a variety of security workflows:
 - Security Research – Deploying vulnerable systems to evaluate new attacker methods and tools
 - Citrix Application Delivery Controller Exploit CVE-2019-19781 as an example.
 - Business Operations – Deploy decoys around actual systems for attackers to target
 - Requires the ability to manage the decoys and their log output
 - Normally a higher maturity level for a security organization, and not the first tool to be deployed.*

What is a Honeypot?

- Honeypots can simulate a client or a server, providing the appropriate behavior depending on the configuration of the honeypot.
 - Client send a specific request that the Server interprets correctly
 - Examples: AD Client, Web Browser.
 - Server sends a specific response that the Client interprets correctly
 - Examples: Web Server, Database.

What is a Honeytrap?

- Types of Honeytraps (categories):
 - Low Interaction
 - Basic Protocol/Application Support without additional interaction.
 - Medium Interaction
 - Protocol/Application Support with the ability to simulate more behavior closer to the actual Protocol/Application.
 - High Interaction
 - Similar to a real system responding with the same Protocol/Application behavior as the real system.

History of Honeypots

- Concepts of a honeypot (in computing)
 - The Cuckoo's Egg – Clifford Stoll (1989)
 - First-hand account of a computer intrusion at the Lawrence Berkeley National Laboratory.
 - Stoll utilized a fake department as a honeypot to catch the attacker, who was spying for the Soviet Union.
 - The Deception Toolkit – Fred Cohen (1997)
 - <http://all.net/dtk/> - includes references and studies on the use of deception.
 - Simulates network services, responds to attackers with an appropriate or deceptive response.
 - Provides a logging mechanism.

History of Honeypots

- honeyd
 - Created by Niels Provos
 - Additionally worked on OpenBSD, OpenSSH, systrace and work on privilege separation
 - Version 0.5 in 2003 (originally released in 2002).
 - honeyd provides network stack behavior for each configured honeypot.
 - Scripted configuration to simulate Windows, Linux, BSD.

History of Honeypots

- Historical perspective
 - Systems in 2020 are mostly virtualized compared to requiring physical systems in the early 2000s.
 - On-Prem or Cloud based architectures make deploying honeypots or decoy systems easier than ever before.
 - Random ports being open or a number of systems sitting out on the Internet is now “the new normal”
 - Think Internet of Things (or Internet of Trash you pick) with billions of devices on the Internet.



History of Honeypots

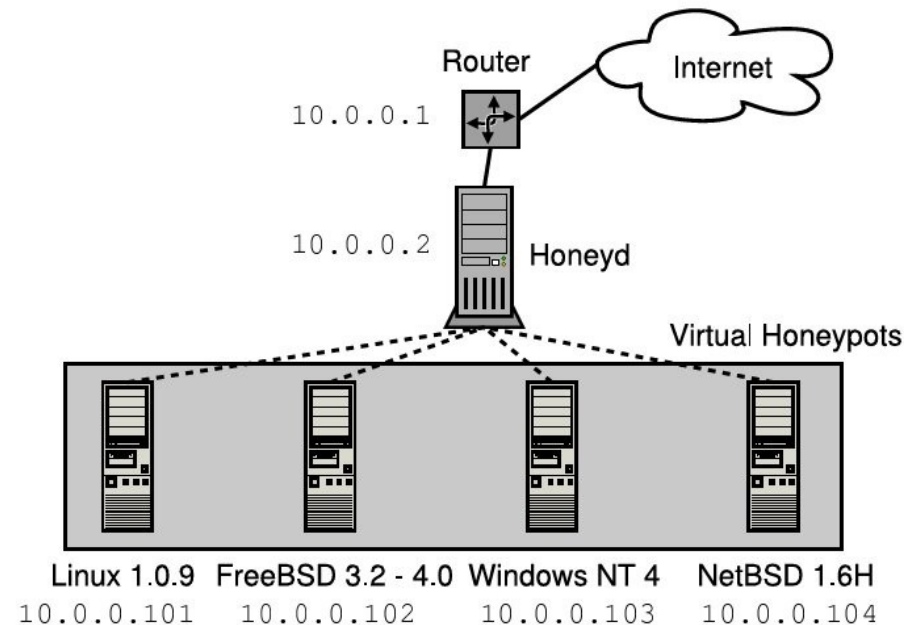


Figure 1: Honeyd receives traffic for its virtual honeypots via a router or Proxy ARP. For each honeypot, Honeyd can simulate the network stack behavior of a different operating system.

(Provos, 2003, p. 3)

Honeypots on BSD

- honeyd
 - Simulates the network stacks using the NMAP OS Fingerprint database
 - Allows for complex network setups simulating various routes including tarpits, responding to unused IP address space to capture traffic.
 - There is also LaBrea which is a sticky honeypot to capture and thwart worms/attackers.

Honeypots on BSD

- honeyd issues on BSD
 - A quick install of the port on FreeBSD/HardenedBSD CURRENT and it did not work, works fine on OpenBSD*
 - Also the OS fingerprints are dated at this point*
 - A note about honeyd later*

Honeypots on BSD

- Basic Setup and Config on OpenBSD

```
# pkg_add honeyd
# cat << EOF > /root/honeyd.conf
create default
set default personality "Linux 2.2.14"
set default default tcp action block
add default tcp port 8080 "/usr/local/share/honeyd/scripts/web.sh"
set default ethernet "00:55:66:ab:8c:22"
dhcp default on vio0
EOF
# honeyd -f /root/honeyd.conf -d
```

Honeypots on BSD

- Output of honeyd while scanning with nmap

```
Honeyd V1.5c Copyright (c) 2002-2007 Niels Provos
honeyd[31660]: started with -f /root/honeyd.conf -d
honeyd[31660]: listening promiscuously on vio0: (arp or ip proto 47 or (udp and src port 67 and dst port 68) or (ip )) and not ether src
00:a0:98:05:b4:6a
honeyd[31660]: HTTP server listening on 127.0.0.1:80
honeyd[31660]: HTTP server root at /usr/local/share/honeyd/webserver/htdocs
honeyd[31660]: [vio0] trying DHCP
honeyd[31660]: Demoting process privileges to uid 546, gid 546
honeyd[31660]: [vio0] got DHCP offer: 172.16.66.143
honeyd[31660]: Updating ARP binding: 00:55:66:1d:b0:f4 -> 172.16.66.143
honeyd[31660]: arp reply 172.16.66.143 is-at 00:55:66:1d:b0:f4
honeyd[31660]: Sending ICMP Echo Reply: 172.16.66.143 -> 172.16.66.6
honeyd[31660]: arp_send: who-has 172.16.66.6 tell 172.16.66.143
honeyd[31660]: arp_recv_cb: 172.16.66.6 at 58:9c:fc:00:01:04
honeyd[31660]: arp_recv_cb: 172.16.66.6 at 58:9c:fc:10:ff:ad
honeyd[31660]: Connection request: tcp (172.16.66.6:35449 - 172.16.66.143:8080)
honeyd[31660]: Connection request: tcp (172.16.66.6:35450 - 172.16.66.143:8080)
honeyd[31660]: Sending ICMP Echo Reply: 172.16.66.143 -> 172.16.66.6
honeyd[31660]: Sending ICMP Echo Reply: 172.16.66.143 -> 172.16.66.6
honeyd[31660]: Connection to closed port: udp (172.16.66.6:61027 - 172.16.66.143:42465)
honeyd[31660]: Sending ICMP Echo Reply: 172.16.66.143 -> 172.16.66.6
honeyd[31660]: Sending ICMP Echo Reply: 172.16.66.143 -> 172.16.66.6
honeyd[31660]: Connection to closed port: udp (172.16.66.6:61027 - 172.16.66.143:42465)
honeyd[31660]: Sending ICMP Echo Reply: 172.16.66.143 -> 172.16.66.6
```

Honeypots on BSD

- Honeytrap - <https://docs.honeytrap.io/> (DutchSec)
 - “HoneyTrap is a modular framework for running, monitoring and managing honeypots. Using HoneyTrap you can use sensors, high interaction and low interaction honeypots together, while still using the same event mechanisms. HoneyTrap consists of services, directors, listeners and channels”
 - Available on FreeBSD/HardenedBSD though mostly Linux focused
 - Will look into a port for OpenBSD as it should work being based on Go*
 - A large number of features, and technically what I should have used in my case study, but more on that later.



Honeypots on BSD

- Default honeytrap.toml configuration file:

```
[listener]  
type="socket"
```

```
[service.ssh-simulator]  
type="ssh-simulator"  
credentials=["root:root", "root:password"]
```

```
[[port]]  
ports=["tcp/8022"]  
services=["ssh-simulator"]
```

```
[channel.console]  
type="console"
```

```
[[filter]]  
channel=["console"]
```

```
[[logging]]  
output="/var/log/honeytrap/logging.log"  
level="debug"
```

Honeypots on BSD

- Logging into the honeytrap ssh service:

```
$ ssh -p8022 root@127.0.0.1
root@127.0.0.1's password:
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-31-generic x86_64)
```

```
* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage
```

```
524 packages can be updated.
270 updates are security updates.
```

```
-----
Ubuntu 16.04.1 LTS                               built 2016-12-10
-----
```

```
last login: Sun Nov 19 19:40:44 2017 from 172.16.84.1
root@host:~$ cat /etc/shadow
cat /etc/shadow: command not found
root@host:~$ ls /etc
ls /etc: command not found
root@host:~$
```

Honeypots on BSD

- Output from interacting with the ssh service:

[illegible]

```
Honeytrap starting (br5vvi8ci1bdqus38er0)...
```

Version: DEVELOPMENT.GOGET (DEVELOPMENT.)

```
services > ssh > category=ssh, date=2020-05-25 13:29:31.047929485 -0400 EDT m=+20.706280804, destination-ip=127.0.0.1, destination-port=8022, sensor=services, source-ip=127.0.0.1, source-port=38054, ssh.publickey-type=ssh-rsa, ssh.publickey=XXXXXXXXXXXXXXXXXXXX, ssh.sessionid=br600uoci1b3dbrt8rtq, ssh.username=root, token=br5vvi8ci1bdqus38er0, type=publickey-authentication
```

```
services > ssh > category=ssh, date=2020-05-25 13:29:31.048095576 -0400 EDT m=+20.706446895, destination-ip=127.0.0.1, destination-port=8022, sensor=services, source-ip=127.0.0.1, source-port=38054, ssh.publickey-type=ssh-ed25519, ssh.publickey=XXXXXXXXXXXXXXXXXXXX, ssh.sessionid=br600uoci1b3dbrt8rtg, ssh.username=root, token=br5vvi8ci1bdqus38er0, type=publickey-authentication
```

```
services > ssh > category=ssh, date=2020-05-25 13:29:34.301457028 -0400 EDT m=+23.959808444, destination-ip=127.0.0.1, destination-port=8022, sensor=services, source-ip=127.0.0.1, source-port=38054, ssh.password=root, ssh.sessionid=br600uoci1b3dbrt8rtg, ssh.username=root, token=br5vvi8ci1bdqus38er0, type=password-authentication
```

```
services > ssh > category=ssh, date=2020-05-25 13:29:34.303881027 -0400 EDT m=+23.962232432, destination-ip=127.0.0.1, destination-  
port=8022, sensor=services, source-ip=127.0.0.1, source-port=38054, ssh.payload=[]byte{0x0, 0x0, 0x0, 0xf, 0x73, 0x63, 0x72, 0x65,  
(a long byte array for the ssh payload)
```

```
0x1, 0x5c, 0x0, 0x0, 0x0, 0x0, 0x5d, 0x0, 0x0, 0x0, 0x0, 0x0}, ssh.request-type=pty-req, ssh.sessionid=br600uoci1b3dbrt8rtg, token=br5vvi8ci1bdqus38er0, type=ssh-request
```

```
services > ssh > category=ssh, date=2020-05-25 13:29:34.304286933 -0400 EDT m=+23.962638332, destination-ip=127.0.0.1, destination-port=8022, sensor=services, source-ip=127.0.0.1, source-port=38054, ssh.payload=[]byte{}, ssh.request-type=shell, ssh.sessionid=br600uoci1b3dbrt8rtg, token=br5vvi8ci1bdqus38er0, type=ssh-request
```

```
services > ssh > category=ssh, date=2020-05-25 13:29:40.095868114 -0400 EDT m=+29.754219428, destination-ip=127.0.0.1, destination-port=8022, sensor=services, source-ip=127.0.0.1, source-port=38054, ssh.command=cat /etc/shadow, ssh.sessionid=br600uoci1b3dbrt8rtg, token=br5vvi8ci1bdqus38er0, type=ssh-channel
```

```
honeytrap > heartbeat > category=heartbeat, date=2020-05-25 13:29:40.362960063 -0400 EDT m=+30.021311476, sensor=honeytrap,
sequence=0, token=br5vvi8ci1bdqus38er0, type=info
```

```
services > ssh > category=ssh, date=2020-05-25 13:29:58.889351221 -0400 EDT m=+48.547702613, destination-ip=127.0.0.1, destination-port=8022, sensor=services, source-ip=127.0.0.1, source-port=38054, ssh.command=ls /etc, ssh.sessionid=br600uoci1b3dbrt8rtg, token=br5vvi8ci1bdqus38er0, type=ssh-channel
```

Honeypots on BSD

- Kippo - <https://github.com/desaster/kippo>
 - Medium Interaction SSH Honeypot
 - Port/Package available on FreeBSD/HardenedBSD, and should work fine on OpenBSD
 - Works with Python 2.5+ and the Twisted framework.
 - Python will be a reoccurring theme for simulating applications and services.
 - Provides a simulated file-system to capture files
 - This project has been forked and is now Cowrie

Honeypots on BSD

- Cowrie - <https://www.cowrie.org/>
 - Medium Interaction SSH/Telnet Honeypot
 - Port/Package available on FreeBSD/HardenedBSD, and should work fine on OpenBSD
 - Works with Python 3.6+ and the Twisted framework.
 - As a fork of Kippo, it provides the same functionality for SSH but with enhancements for SSH and Telnet proxying to observe attackers behavior.

Honeypots on BSD

- Beyond what is available as a port or package in the various BSD operating systems, there are numerous honeypots available for a variety of specific applications and protocols.
- Think about the complexity of DCERPC/SMB from Microsoft, luckily others have attempted to mimic these protocols.

Honeypots on BSD

- Awesome Honeypot list:
<https://github.com/paralax/awesome-honeypots>
- “A curated list of awesome honeypots, plus related components and much more, divided into categories such as Web, services, and others, with a focus on free and open source projects.”
- Roughly ~280 links to explore

Honeypots on BSD

- From this list of honeypots, I selected one I thought would fit my needs, as I just wanted something simple
 - Turns out, when you just jump into somethings, you get more involved than you originally wanted.
 - But the solution works in the end so you are content with the result.



Honeypots on BSD

- Of the available applications, I selected HoneyPy


<https://github.com/foospidy/HoneyPy>

- Onto the reason I started messing with honeypots.

Case Study of a Honeypot on FreeBSD

- Story begins with a requirement to collect threat data:
 - The company I was working with required a method to produce threat data to share with its customers.
 - At the time, there was no security research of this type occurring at the company.
 - Dark nets, dirty connections, anything that was fun* was not available.
 - We needed a system to be configured to monitor for potential threats and be able to share this threat data with our customers.

Case Study of a Honeypot on FreeBSD

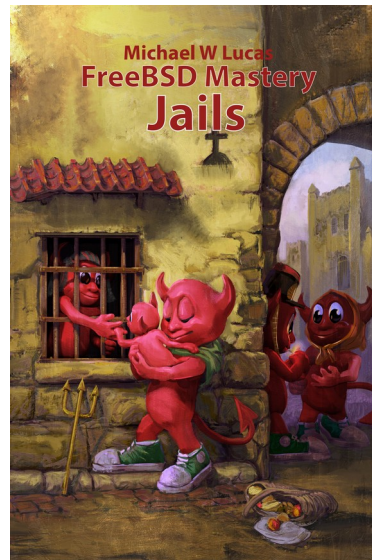
- Of course this will run on BSD, why not a FreeBSD Jail?
 - During the initial Mirai infections in 2016/2017, I enabled the telnetd service in a FreeBSD jail with inetd.conf to start on connections.
 - Technically a honeypot, but more of a deceptive service.*
 - • If you do not know what inetd is, go read a UNIX book about the good ole days of preserving as much of your system resources as possible (The inetd utility appeared in 4.3BSD.)
 - Not only could I setup a jail as a honeypot, but use another jail to monitor the traffic from the honeypot
 - Using Zeek or Snort/Suricata and correlate with available threat data

Case Study of a Honeypot on FreeBSD

- Design of the system - FreeBSD Host setup with an Internet connection.
 - I used Digital Ocean, but any cloud provider would do that allows console access and FreeBSD.
 - Received a single IP address, but FreeBSD jails require an IP to be configured (More on this in a second)
 - Jail management – I selected iocage, which was my first time using it and served my needs.

Case Study of a Honeypot on FreeBSD

- Installation and setup of iocage
 - Please see Michael W. Lucas's FreeBSD Jails Mastery for complete features of iocage and other jail management tools:
 - <https://www.tiltedwindmillpress.com/product/fmjail/>



Case Study of a Honeypot on FreeBSD

- Example configuration for pf to redirect traffic:

```
ext_if="vtnet0"  
int_if="lo0"
```

```
#sshguard  
table <sshguard> persist  
HELL_NET="127.6.66.0/24"
```

```
HONEYPY="127.6.66.1"  
ZEEK="127.6.66.2"
```

```
set block-policy drop  
set optimization aggressive  
scrub in on $ext_if all fragment reassemble min-ttl 15 max-mss 1400 no-df
```

```
#jail firewall  
nat pass on $ext_if from $HELL_NET to any -> ($ext_if)  
rdr pass on $ext_if proto tcp from any to any port 22 -> $HONEYPY port  
10022
```

Case Study of a Honey pot on FreeBSD

- In order to setup the honeypot for SSH connections, I am redirecting the traffic to the HoneyPy jail. You need a “rdr” rule for every service configured in your honeypot

```
nat pass on $ext_if from $HELL_NET to any -> ($ext_if)
```

```
rdr pass on $ext_if proto tcp from any to any port 22 -> $HONEYPY port 10022
```

- Ensure the following is setup for each jail on the local interface with an alias in /etc/rc.conf

```
ifconfig_lo0_alias0="inet 127.6.66.1 netmask 0xffffffff"
```

```
ifconfig_lo0_alias1="inet 127.6.66.2 netmask 0xffffffff"
```

Case Study of a Honeyypot on FreeBSD

- Ensure ip forwarding and pf are enabled to start on boot in rc.conf, and reboot to ensure all updates are working before moving on

```
# sysrc gateway_enable=yes
```

```
# sysrc pf_enable=yes
```

```
# reboot
```

Case Study of a Honeypot on FreeBSD

- Basic setup of two jails with iocage (assumes ZFS is available and zroot exists as a zpool):

```
# pkg install -y py37-iocage
```

```
# iocage activate zroot
```

(zroot is the zfs dataset on my VM, this sets up a structure for iocage to use for created jails)

```
# iocage fetch -r 12.1-RELEASE
```

(select a supported FreeBSD release, I used 12.1-FreeBSD)

```
# iocage create -n HoneyPy ip4_addr="127.6.66.1" -r 12.1-RELEASE
```

```
# iocage create -n zeek ip4_addr="127.6.66.2" -r 12.1-RELEASE
```

```
# iocage start HoneyPy
```

```
# iocage set boot=on HoneyPy
```

```
# iocage start zeek
```

```
# iocage set boot=on zeek
```

```
# sysrc iocage_enable=yes
```

Case Study of a HoneyPot on FreeBSD

- Install the packages in the jails:

```
# iocage pkg HoneyPy update
# iocage pkg HoneyPy install -y python2 py27-twisted
# iocage pkg zeek update
# iocage pkg zeek install -y zeek
```

- Setup HoneyPy Jail:

```
# fetch https://github.com/foospidy/HoneyPy/archive/master.zip
# unzip master.zip
# mv HoneyPy-master /usr/iocage/jails/HoneyPy/root/root/
(or wherever your jails reside, place in the root folder)
```

Case Study of a Honeyypot on FreeBSD

- Out-of-the-box services can be enabled in the configuration file located here
`/root/HoneyPy-master/etc/services.cfg` (within the jail)
- By default, all of the output will log to the relative log directory here
`/root/root/HoneyPy-master/log` (within the jail)
- A service can be configured with any of the available plugins, a simple low-interaction setup is to just echo back what is sent.

Case Study of a Honey pot on FreeBSD

- There is an issue with the default services running, so make a backup of that file and start fresh with the services you want.

```
cp /root/HoneyPy-master/etc/services.cfg  
/root/HoneyPy-master/etc/services-back
```

- The following is an example of using the Echo plugin to just echo SSH traffic back when received by the service.

```
cat << EOF > /root/HoneyPy-master/etc/services.cfg;  
[SSH]  
plugin      = Echo  
low_port    = tcp:22  
port        = tcp:10022  
description = SSH/SCP on port 22  
enabled     = Yes  
EOF
```


Case Study of a HoneyPot on FreeBSD

- Note that HoneyPy when started will listen on TCP port 10022, but expects that TCP port 22 would be used for the connection.
- pf, the packet filter firewall will handle the redirection of traffic for us as configured on the FreeBSD host.
 - All TCP port 22 traffic will be redirected to TCP port 10022. HoneyPy references ipt-kit to perform the redirection, but pf will do this for you.
- HoneyPy can be started in daemon-mode when running in production, or can be run in a console mode (next slide) providing access to stop and start the services.
 - Running the command from inside the jail

```
# /usr/local/bin/python2.7 Honey.py -d
```
 - From outside the jail with iocage

```
# iocage exec HoneyPy "/usr/local/bin/python2.7 /root/HoneyPy-master/Honey.py -d &"
```

Case Study of a Honeypot on FreeBSD

```
/usr/local/bin/python2.7 Honey.py
```

Your service configuration suggests that you want to run on at least one low port!

To enable port redirection run the following ipt-kit (<https://github.com/foospidy/ipt-kit>) commands as root:

[HoneyPy Copyright (c) 2013-2017. foospidy]

HoneyPy Console. For help type 'help'.

```
HoneyPy> start
```

```
16 service(s) started!
```

```
HoneyPy> list
```

```
FTP      <<class 'twisted.internet.tcp.Port'> of plugins.Echo.Echo.pluginFactory on 10021>
```

```
SSH      <<class 'twisted.internet.tcp.Port'> of plugins.Echo.Echo.pluginFactory on 10022>
```

```
Telnet <<class 'twisted.internet.tcp.Port'> of plugins.Echo.Echo.pluginFactory on 10023>
```

```
SMTP      <<class 'twisted.internet.tcp.Port'> of plugins.Smtplib.Smtplib.pluginFactory on 10025>
```

```
DNS <plugins.DnsUdp.DnsUdp.pluginMain on 10053>
```

```
HTTP <<class 'twisted.internet.tcp.Port'> of plugins.Web.Web.pluginFactory on 10080>
```

```
HTTPS <<class 'twisted.internet.tcp.Port'> of plugins.Web.Web.pluginFactory on 10443>
```

```
DCERPC <<class 'twisted.internet.tcp.Port'> of plugins.Echo.Echo.pluginFactory on 10135>
```

```
DCERPC2 <<class 'twisted.internet.tcp.Port'> of plugins.Echo.Echo.pluginFactory on 10137>
```

```
DCERPC3 <<class 'twisted.internet.tcp.Port'> of plugins.Echo.Echo.pluginFactory on 10139>
```

```
CIFS    <<class 'twisted.internet.tcp.Port'> of plugins.Echo.Echo.pluginFactory on 10445>
```

```
CIFS.udp <plugins.Echo_udp.Echo.pluginMain on 10445>
```

```
MSSQL <<class 'twisted.internet.tcp.Port'> of plugins.Echo.Echo.pluginFactory on 1434>
```

```
RDP      <<class 'twisted.internet.tcp.Port'> of plugins.Echo.Echo.pluginFactory on 3389>
```

```
MySQL <<class 'twisted.internet.tcp.Port'> of plugins.Echo.Echo.pluginFactory on 3306>
```

```
Elasticsearch <<class 'twisted.internet.tcp.Port'> of plugins.Elasticsearch.elasticsearch.pluginFactory on 9200>
```

HoneyPy>

Case Study of a Honeypot on FreeBSD

- Example log event
- 2020-05-15 00:02:36,467784,+0000
[Echo,97469,XX.XX.XX.XX] 620669b0-963f-
11ea-af87-2a017b7d7b0b TCP RX 127.6.66.1
10022 SSH XX.XX.XX.XX 33962
5353482d322e302d50555454590d0a
- Hex string == SSH-2.0-PUTTY

Case Study of a Honeypot on FreeBSD

- HoneyPy jail will have services started for all of the emulated services, while the Zeek jail will monitor all of the traffic.
- In order for the jail to monitor the host interface, a devfs rule must be configured.
- Create the following in /etc/devfs.rules on the FreeBSD host

```
[devfsrules_jail_bpf=7]  
add include $devfsrules_jail  
add path 'bpf*' unhide
```

Case Study of a Honeypot on FreeBSD

- Add the following to the `/etc/rc.conf` on the FreeBSD host:

```
devfs_system_ruleset="devfsrules_jail_bpf"
```

- Using the `iocage "set"` command, ensure the zeek jail loads this devfs ruleset, which gives access to sniff packets from within the jail, then restart devfs and the jail:

```
# iocage set devfs_ruleset=7 zeek
```

```
# service devfs restart
```

```
# iocage restart zeek
```

- Now we can configure the Zeek jail.

Case Study of a Honeypot on FreeBSD

- Zeek NSM for Intel Monitoring and Traffic recording – <https://www.zeek.org>
 - Zeek provides a simple way to ingest threat data and generate notice logs when a specific IOC is observed.
 - Anything else captured that was protocol specific would be logged by Zeek.
 - Suricata can also be used with IP/URL/HASH signatures, but I am more used to having Zeek just work with the intel data*

Case Study of a Honeypot on FreeBSD

- The goal was to combine various OS INT threat sources to create a threat feed for Zeek
- A script automatically downloads the CSV threat data and puts it into an intel.dat file Zeek can read in
 - There are steps to setup MISP on FreeBSD, and I am hoping to be able to get a port setup which would allow the aggregation of multiple threat sources and provide this intel output, instead of a perl script*.

Case Study of a Honeypot on FreeBSD

- Zeek intel configuration example (Note: this is only two lines in the file, made larger for this slide, and all space is **TABS** separating fields):

```
#fields indicator          indicator_type  meta.source
      meta.url            meta.do_notice

006b18b6fbea18750367f1753d9a418d23983823a01bea3ffdbc
551eeced97fd          Intel::FILE_HASH
AlienVault OTXv2 - Signed Executable Potentially
Related to TrickBot and Lazarus Group Activity ID:
5df94019452f666b340101d7 Author: AlienVault
https://twitter.com/VK_Intel/status/12064979098580
78720 T
```


Case Study of a Honeytrap on FreeBSD

- Zeek indicator types (Zeek 3.0.6):

```
## Enum type to represent various types of intelligence data.
type Type: enum {
  ## An IP address.
  ADDR,
  ## A subnet in CIDR notation.
  SUBNET,
  ## A complete URL without the prefix ``"http://"``.
  URL,
  ## Software name.
  SOFTWARE,
  ## Email address.
  EMAIL,
  ## DNS domain name.
  DOMAIN,
  ## A user name.
  USER_NAME,
  ## Certificate SHA-1 hash.
  CERT_HASH,
  ## Public key MD5 hash. (SSH server host keys are a good example.)
  PUBKEY_HASH,      };
```

Case Study of a Honeypot on FreeBSD

- Zeek config updates to load a formatted intel file (add to /usr/local/share/zeek/site/local.zeek or create your own file and load it with local.zeek and load it within the zeek jail):

```
@load policy/frameworks/intel/seen
```

```
@load policy/frameworks/intel/do_notice
```

```
@load policy/frameworks/files/hash-all-files
```

```
redef Intel::read_files += {  
    "/nsm/intel/intel.dat",  
};
```

Case Study of a Honeypot on FreeBSD

- Ensure a properly formatted intel file exists at `/nsm/intel/intel.dat`
- Run the following to enable zeek to start on jail startup

```
# iocage exec zeek "sysrc zeek_enable=yes"
```

Case Study of a Honeypot on FreeBSD

- Now run the zeek deploy command from within the jail
 - There are numerous other configurations for Zeek, I am only covering the basic items to get things running.

```
# iocage exec zeek "sed -i ' ' -e 's/localhost/  
127\.6\.66\.2/g' /usr/local/etc/node.cfg"
```

```
# iocage exec zeek "sed -i ' ' -e  
's/eth0/vtnet0/g' /usr/local/etc/node.cfg"
```

```
# iocage exec zeek zeekctl deploy
```

Case Study of a Honeypot on FreeBSD

- Now with HoneyPy running fake services, and a list of IOCs to monitor for, we can correlate honeypot hits to intel hits to provide a fidelity that the IOCs are actually actively scanning hosts.
- Knowing this can reduce the time on incident response for scanners compared to real attackers.

Case Study of a Honeypot on FreeBSD

- Correlation of logs from Zeek and HoneyPy showing this IOC is still scanning and active.

```
conn.log:1590637759.244217      Czwhu743bo5qkWgBd4      XX.XX.XX.XX
52080  XX.XX.XX.30  22      tcp      -      33.907172      968
968    RSTO      F      T      0      ShADdR  9      1340      4
1140    -
```

```
intel.log:1590637759.244276      Czwhu743bo5qkWgBd4      XX.XX.XX.XX
52080  XX.XX.XX.30  22      XX.XX.XX.XX  Intel::ADDR
Conn::IN_ORIG  zeek      Intel::ADDR
Emerging_Threats_Compromised_IPs  -      -      -
```

```
ssh.log:1590637760.196914      Czwhu743bo5qkWgBd4      XX.XX.XX.XX
52080  XX.XX.XX.30  22      2      -      0      INBOUND SSH-
2.0-paramiko_1.7.5  SSH-2.0-paramiko_1.7.5  aes128-ctr      hmac-
sha2-256      none      diffie-hellman-group-exchange-sha256      ssh-
rsa
```

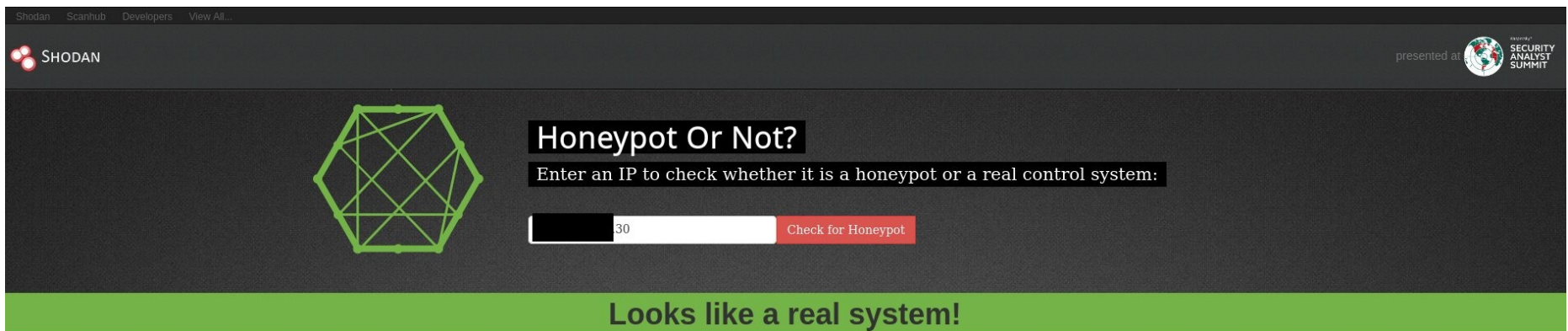
```
HoneyPy.log:2020-05-28 03:49:20,197160,+0000 [Echo,29,XX.XX.XX.XX]
35a664b0-a096-11ea-9769-2a017b7d7b0b TCP RX 127.6.66.1 10022 SSH
XX.XX.XX.XX 52080 5353482d322e302d706172616d696b6f5f312e372e350d0a
```

Case Study of a Honeypot on FreeBSD

- I have scripts I created to correlate and provide this information to the customers.
- My case study ends with this initial setup, but there are other directions to take this build-out.
- The good news is Shodan has an API to evaluate whether a system is real or looks like a honeypot.

Case Study of a Honeypot on FreeBSD

- So far, my system appears to be legitimate, not a honeypot
- Without additional analysis, it looks like a FreeBSD firewall redirecting traffic, not a cloud instance hosting a honeypot jail.



Case Study of a Honeypot on FreeBSD

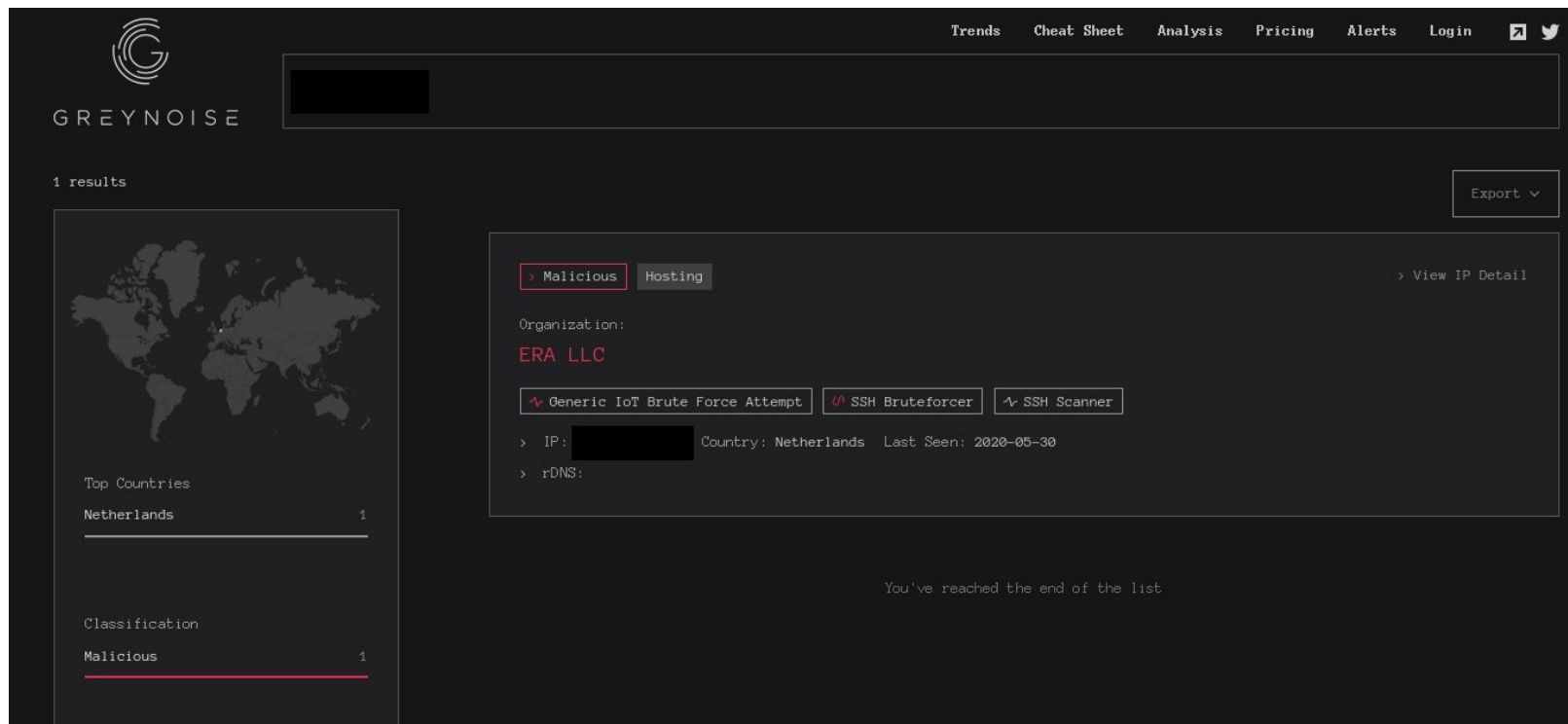
- Getting more from this data:
 - Validate against the customer log data
 - Are they seeing any of the correlated honeypot/zeek IOCs in their own logs? Are the IOCs still valid?
 - MISP provides a mechanism to aggregate this data and mark specific IOCs as being too old, or aging out, and automating the update of the threat data.
 - Zeek supports updates to the intel files and continues to run.
 - This is part of the way the input framework works within Zeek.

Case Study of a Honeypot on FreeBSD

- Getting more from this data:
 - Configuring API integration to validate if the traffic is a known scanner or more of a potential threat.
 - GreyNoise provides this service to identify known scanners or potentially malicious traffic:
<https://greynoise.io>
 - If a system has been observed by GreyNoise's sensors, they will add additional context to the alerts.

Case Study of a Honeypot on FreeBSD

- GreyNoise lookup for the IP in this alert



Case Study of a Honeypot on FreeBSD

- This case study shows but one example of setting up a Honeypot on BSD operating systems.
- Outside of this collected data in simple log formats, aggregating this log data and looking for trends can provide further insight into potential new trends in attacker behavior.
- Taking this a step further is to ingest this data into a Security Incident and Event Management (SIEM) system to evaluate threat data to actual traffic in your environment.
 - The ability to ask “I saw this attack hit the honeypot, from this specific source host, did this source host access anything else?”

Recent Examples of Honeypots in Security Research

- Vulnerability in Citrix Application Delivery Controller CVE-2019-19781

“An issue was discovered in Citrix Application Delivery Controller (ADC) and Gateway 10.5, 11.1, 12.0, 12.1, and 13.0. They allow Directory Traversal.” (NIST, 2020)

- Runs FreeBSD under the hood.

Recent Examples of Honeypots in Security Research

- Details provided by US Cert (2020) show an example of successful exploitation of this vulnerability:

```
10.1.1.1 - - [10/Jan/2020:13:23:51 +0000] "POST
/vpn/./vpns/portal/scripts/newbm.pl HTTP/1.1"
200 143 "https://10.1.1.2/" "USERAGENT "
```

```
10.1.1.1 - - [10/Jan/2020:13:23:53 +0000]
"GET /vpn/./vpns/portal/backdoor.xml HTTP/1.1"
200 941 "-" "USERAGENT"
```

Recent Examples of Honeypots in Security Research

- Building upon this example, a honeypot can be created to look for the HTTP POST request, respond with an HTTP 200 OK and perform further analysis
- MalwareTech created a honeypot with python3 and openssl

<https://github.com/MalwareTech/CitrixHoneypot>

Recent Examples of Honeypots in Security Research

Lines 44-48 show that with the honeypot configured using SSL on TCP port 443, it looks for the attempt to traverse the directory before continuing to monitor the request

```
# only proceed if a directory traversal was attempted
if path.find('/../') != -1:
    # flatten path to ease parsing
    collapsed_path = server._url_collapse_path(path)
    url_path = list(filter(None, collapsed_path.split('/')))
```


Recent Examples of Honeypots in Security Research

- If an attacker sends a request to one of the other pages, the python code will load a separate HTML to mimic the real application, such as the login page.
- There is also a funny response, if a scanner or attacker sends an exploit attempt but the HTTP client sanitizes the traversal “/../” path, they get a gold star.

Recent Examples of Honeypots in Security Research

- This is an example of a critical vulnerability with detection details turning into the creation of a specific honeypot to catch exploit attempts
- Honeypots deployed can help to further the research about these attempts, or to provide companies running Citrix NetScaler/ADC that someone may be targeting them, but luckily only hit the honeypot
- With Python3 and SSL, this honeypot can be setup on BSD operating systems.

Conclusion

- BSD Operating System are well suited for honeypots
 - And are a part of the history of honeypots
- I will be working to get the FreeBSD/HardenedBSD port fixed for honeyd, and to get the honeytrap port created for OpenBSD.
 - Hoping to bump honeyd version itself to support the NMAP Fingerprints of today.
- I am going to spend some time reworking what I did in my setup to use honeytrap, as it is an interesting way to handle multiple services.
 - One step crazier is setup a specific service per jail

Conclusion

- If anything from this talk, hopefully more honeypots will start using BSD operating systems over docker images.



Thank you

References

- Cohen, F. (1997). The Decption Toolkit. Retrieved from <http://all.net/dtk/>.
- Cowrie. (2020). Cowrie SSH/Telnet honeypot. Retrieved from <https://www.cowrie.org/>
- Desaster. (2020). Kippo SSH honeypot. Retrieved from <https://github.com/desaster/kippo>
- Foospidy. (2020). HoneyPy honeypot. Retrieved from <https://github.com/foospidy/HoneyPy>
- GreyNoise. (2020). GreyNoise Visualizer. Retrieved from <https://greynoise.io>
- HoneyTrap. (2020). HoneyTrap. Retreied from <https://docs.honeytrap.io/>
- MalwareTech. (2020). CitrixHoneypot. Retrieved from <https://github.com/MalwareTech/CitrixHoneypot>
- NIST. (2020). National Vulnerability Database – CVE-2019-19781. Retrieved from <https://nvd.nist.gov/vuln/detail/CVE-2019-19781>

References

- Parallax. (2020). Awesome honeypots. Retrieved from <https://github.com/parallax/awesome-honeypots>
- Provos, N. (2003). A virtual honeypot framework. Retrieved from <http://www.citi.umich.edu/u/provos/papers/honeyd.pdf>
- Shodan. (2020). Honeyscore. Retrieved from <https://honeyscore.shodan.io/>
- US Cert. (2020). Detecting Citrix CVE-2019-19781 – Alert (AA20-031A). Retrieved from <https://www.us-cert.gov/ncas/alerts/aa20-031a>
- Wikipedia. (2020). The Cuckoo's Egg. Retrieved from https://en.wikipedia.org/wiki/The_Cuckoo's_Egg
- Zeek. (2020). Zeek NSM. Retrieved from <https://www.zeek.org>

Questions?

Copyright (c) 2020, Michael Shirk, Daemon Security Inc.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.