A Security Framework for IPv6 Deployment

ULAKBIM

January 17, 2011

Contents

Chapter 1

About

1.1 What is Kovan?

Kovan is an IPv6 honeypot project. It is developed under the Design of National IPv6 Infrastructure and Transition to IPv6 Protocol project in order to

- Analyze of IPv6 transition mechanisms from security perspective
- Research on vulnerabilities and threads in IPv6
- Investigate worm propagation in IPv6 networks
- Identify current and new security threads in IPv6 networks

Kovan is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under the Support Programme For Research Projects of Public Institution (KAMAG 1007) and developed by TUBITAK ULAKBIM

1.2 System requirements

Kovan system requirements strictly depend how Kovan is configured. Base recommended hardware configuration for direct installation.

- 2 Gigahertz (GHz) or faster 64-bit (x64) single core processor
- 2 Gigabyte (GB) of RAM

If you install Kovan KVM/QEMU image recommended hardware is

- 2 Gigahertz (GHz) or faster 64-bit (x64) dual or quad core processor
- 4 Gigabyte (GB) of RAM

Hardware requirements for Kovan increase if you add KVM images to your Kovan architecture. As a rule of tomb, the amount of RAM and the number of cpu core should increase as the number of the KVM images increase. To give a baseline, ULAKBIM Kovan test system runs 12 qemu images (1 core, 512 Mbyte for each) withouh any significant hardware penalty.

- 2x Intel Xeon CPU E5520 @2.27 Ghz
- 12 Gbyte of RAM
- 2x 160 Gbyte 15.000 RP SAS HDD

1.3 Licensing

Kovan is available under the GPL for use with other GPL-licensed software and FOSS applications licensed under GPL-compatible FOSS licenses.

Kovan has a commercial license for all of its software that is embedded in or bundled with another application. For more information, please send an email to license@ipv6.net.tr.

1.4 Downloading the latest version

Kovan web address is www.ipv6.net.tr/kovan, you can download the latest verion in dowload section.

Chapter 2

Release Notes

2.1 What's new in this version

This is the first version of Kovan

2.2 Known issues

Kovan is developed on FreeBSD 8.1. As a result, all the install and run scripts and compiled code are tested for 8.1 version. However, thanks to FreeBSD port system architecture, Kovan should be able to run for the Future FreeBSD releases.

Kovan use RIP as an routing protocol and RIP has a maximum hop count 15. Thus, if your virtual network topology has more hop than 15 hops routing fail.

Kovan install scripts use FreeBSD Ports system without `FORCE_PKG_REGISTER`setting, thus if the package is installed before the startup system will fail. If this is the case set the `FORCE_PKG_REGISTER`to 1 (i.e export `FORCE_PKG_REGISTER=1`for sh like shells)

For some topologies, Kovan uses more than one epair instances. When destroying this kind of topology Kovan waits 2 seconds to destroy each epair object. Otherwise, System gives panic error and reboots.

Chapter 3

Basics

3.1 Virtual Honeypot Framework

Kovan is a virtual honeypot framework that mimics a real network behavior to attract attackers. Virtual honeypot idea is feasible for system holders because their system requirement is fewer compared to real systems. In addition, virtual honeypots are more attractive in the view of attackers because a "real looking" network topology can be implemented more easily. Kovan extensively utilizes virtual networking concept to provide a working environment for upper level honeypots such as Argos[1] and Nepenthes[2].

Kovan combines several concepts to build a virtual honeypot framework. In the big picture, Kovan firstly creates virtual devices that will act as hosts, servers and routers. FreeBSD Jails[3] system is used to create virtual devices, section 3.1.1 gives detailed information about virtual devices. On the second step, virtual links are established between suitable devices. FreeBSD's Netgraph[4] and Epair[5] systems are used in linking step and section 3.1.3 and 3.1.2 will cover details. Up to third step, devices are created and linked to each other via links. The third -and the last- step is aims to establish connectivity between network devices. Connectivity is achieved by using dynamic routing via Quagga[6] software, section 3.1.4 contains details about dynamic routing.

3.1.1 Virtual Devices: FreeBSD Jail

The FreeBSD jail system is an OS level virtualization that provides independent, FreeBSD-based, low resource consuming virtual systems. This virtual systems are called jails and can provide lots of the functionality that a real system does.

In the standard jail setup, it is only supported to give IP addresses from the same subnet of host system to jails. In addition, connecting jails each other to create complex networks were not available in standard setup. Kovan uses a more powerful jail configuration for networking operations. This is achived by an jail improvement named VIMAGE, whic is provided with version 8 of FreeBSD. In order to create a virtual network, Kovan creates virtual network interfaces as explained in sections 3.1.2-3.1.3 and links them as wanted, later virtual interfaces are moved to suitable virtual devices(jails). What vimage improvement provides is that moving virtual interfaces into jails . In order to use vimage support FreeBSD kernel must be recompiled (section 5.4.1).

```
1 root# jail -c vnet name=n0 host.hostname=n0 path=/usr/local/kovan/node persist
2 root# ngctl mkpeer eiface ether ether    // ngeth0 is created
3 root# ifconfig ngeth0 vnet n0
4 root# jexec n0 ifconfig ngeth0 name eth0
5 root# jexec n0 ifconfig eth0 link 00:0c:6e:22:13:0
```

Listing 3.1: Creating a virtual interface and assigning it to a jail

Listing 3.1 explains jail vimage support with a simple example. On the first line the jail n0 is created. Second line creates a netgraph object of type eiface, this object can be seen as a ethernet interface, detailed information about netgraph system is given at section 3.1.2. Line three moves new object ngeth0 to jail n0 using ifconfig command. Line 4-5 changes the name of the netgraph interface to a more realistic eth0 and sets the link address. These commands creates virtual computer with a virtual interface at the end. This is not a practical example because virtual interface is not connected to any other interface.

Kovan uses jails to create several network components such as routers, servers and clients. Each network components is belongs to a class and its properties are determined by this class. Kovan has three device classes; router, node and monitor. As the name implies, router class is used for virtual routers. All router required software must be installed on this jail. Monitor jail is used for virtual network monitoring. Net-SNMP, Nagios, NFSEN and MRTG softwares are installed on monitor. Node type jail can be used as server or client devices. One can install any service on a node type jail. Kovan's fake services HTTP, SMTP, FTP and DNS services are installed on node jail.

3.1.2 Virtual Links: Netgraph

The netgraph system provides network objects(nodes) that perform various networking functions. Nodes can be connected using special connectors -named hooks- to create complicated graphs. In Kovan, netgraph nodes are moved into virtual devices and than connected to create a virtual network among virtual devices. In the simplest case, netgraph nodes can be seen as ethernet cards and hooks can be seen as ethernet cables.

Netgragh Nodes used in Kovan system:

- ng_eiface: The eiface netgraph node implements the generic virtual Ethernet interface. When eiface node is created, a new interface appears accessible via ifconfig(8). Kovan uses ng_eiface in two ways; connecting two nodes directly or connecting a node to a bridge.

- ng_bridge: The bridge netgraph node performs ethernet bridging over one or more links. Each link (represented by a connected hook) is used to transmit and receive raw ethernet frames. As packets are received, the node learns which link each host resides on. Packets unicast to a known host are directed out the appropriate link only, and other links are spared the traffic. This behavior is in contrast to a hub, which always forwards every received packet to every other link. Kovan uses bridge node to connect more than two eiface nodes to create local area networks.

- ng_pipe: This node permits simple traffic shaping by emulating bandwidth and delay, as well as random packet losses. The node has two hooks, upper and lower. Traffic flowing from upper to lower hook is referenced as downstream, and vice versa. Parameters for both directions can be set separately, except for delay. Kovan puts pipe nodes to both direct connections and node-to-bridge connections. System administrator can change link parameters to make network more realistic.

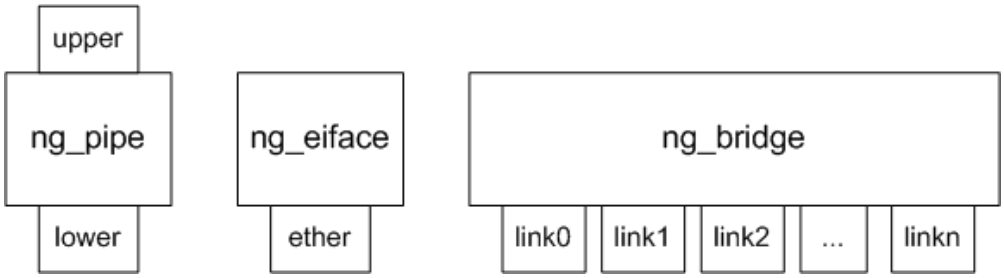

Figure 3.1: Netgraph objects used in Kovan

Figure 3.1 represents netgraph objects used in Kovan which are ng_eiface, ng_bridge and ng_pipe.

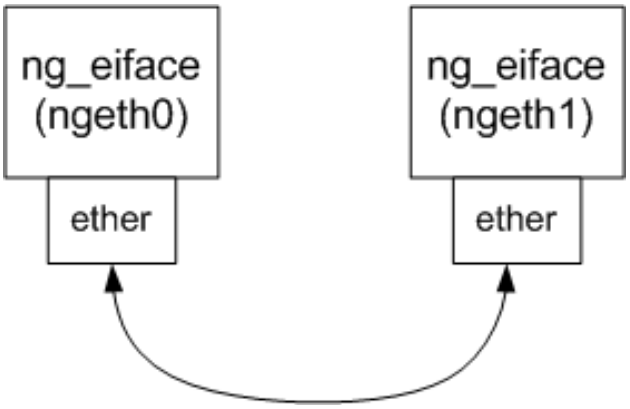

Figure 3.2: Connecting two eiface nodes

Figure 3.2 shows a simple example about netgraph nodes and hooks. In this example two ng_eiface nodes -ngeth0 and ngeth1- are created, each eiface node has a hook named ether. In order to connect two eiface nodes, their ether hooks are connected. This setup can be established via commands represented in listing 3.2. Line 1 and 2 create two ng_eiface nodes named ngeth0 and ngeth1. Node names are assigned automatically by the system but can be changed later. When a ng_eiface node is created it gets the name of form ngethX where X is the current numbers of ng_eiface nodes in the system. Therefore ngeth0 node is created on the first line and ngeth1 is created on the second line. Third line links two previously created nodes. ïngctl connectc̈ommand takes the names of nodes and their hooks to be connected in relative order.

```
1 root# ngctl mkpeer eiface ether ether
2 root# ngctl mkpeer eiface ether ether
3 root# ngctl connect ngeth0: ngeth1: ether ether
```

Listing 3.2: Connecting two ng_eiface node

There is an improved setup in figure 3.3 where two ng_eiface nodes are connect via a ng_pipe object.

Pipe object provides a messaging mechanism to change link properties such as bandwidth, delay, bit error rate (BER) and duplication values. In listing 3.3 commands required to connect two nodes with a pipe object is given. Third line creates a pipe object and connects ngeth0 to pipe's upper hook.

```
ngctl mkpeer [target] type hook1 hook2
```

It is important to note that netgraph objects can not be created alone, while creation you must give a target node to connect. Even "target" field is put empty created node is connected to (default) socket node, as done on the first line of listing 3.3.

```
1  root# ngctl mkpeer eiface ether ether
2  root# ngctl mkpeer eiface ether ether
3  root# ngctl mkpeer ngeth0: pipe ether upper
4  root# ngctl name ngeth0:ether n0-n1
5  root# ngctl connect n0-n1: ngeth1: lower ether
6  root# ngctl msg n0-n1: setcfg { bandwidth=50000 }
7  root# ngctl msg n0-n1: setcfg { delay=10 }
8  root# ngctl msg n0-n1: setcfg { upstream={ BER=2 } }
9  root# ngctl msg n0-n1: setcfg { upstream={ duplicate=1 } }
10 root# ngctl msg n0-n1: setcfg { downstream={ fifo=1 } }
```

Listing 3.3: Connecting two ng_eiface node with pipe

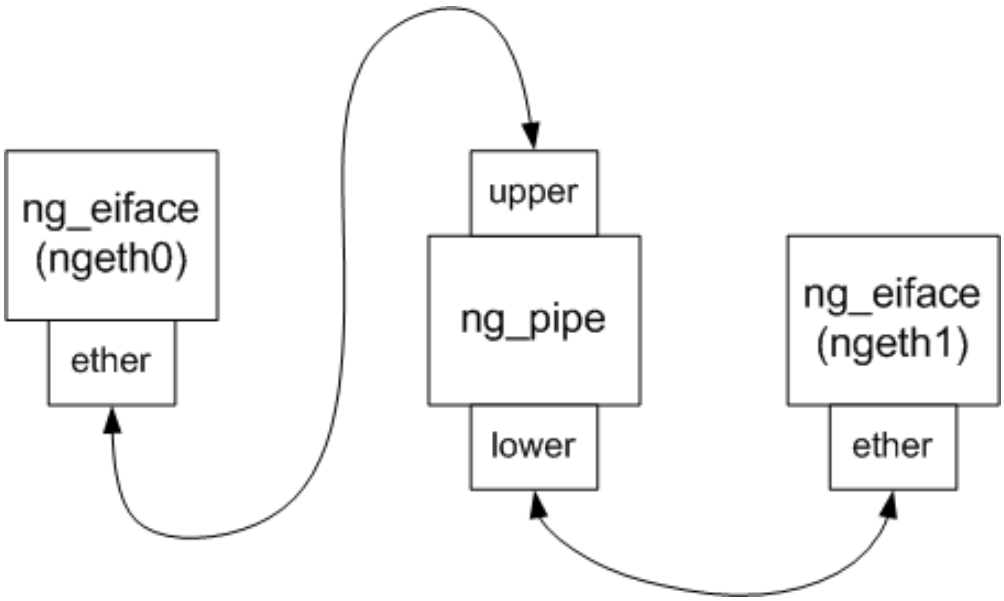

Figure 3.3: Connecting two eiface nodes with pipe

Fourth line of listing 3.3 gives a unique name to pipe object. Some objects like pipe do not get a unique name while created instead name must be set after creation. Line 4 sets n0-n1 name to the object connected to ngeth0's ether hook which is the pipe node created on line three.

Listing 3.4 example creates 4 ngeth object and connects them with a bridge object. Bridge object has several hooks numbered link0, link1, link2 ... linkn. In this example pipe objects are not used for clarity but a pipe object can be used on each connection if wanted.

```
1 root# ngctl mkpeer eiface ether ether
2 root# ngctl mkpeer eiface ether ether
3 root# ngctl mkpeer eiface ether ether
4 root# ngctl mkpeer eiface ether ether
5 root# ngctl mkpeer ngeth0: bridge lower link0
6 root# ngctl name ngeth0:lower test-bridge
7 root# ngctl connect ngeth1: test-bridge: ether link1
8 root# ngctl connect ngeth2: test-bridge: ether link2
9 root# ngctl connect ngeth3: test-bridge: ether link3
```

Listing 3.4: Connecting four eiface nodes via a bridge

These linked netgraph nodes are moved to appropriate device to build a network topology. Moving netgraph objects does not destroy links. When a netgraph eiface is moved to an vimage Jail (section 3.1.1) it will seen among network interfaces.

Netgraph system provides lots of ng_ objects but we use just a subset of them currently. In the future Kovan can be improved by using additional structures

3.1.3 Virtual Links: Epair

Epair system is similar to netgraph system in the way they both used for linking network devices. There are two major differences between epair and netgraph systems:

- Epair system does not have any hook mechanism or other complicated connection routines. Instead, epair object are easily connected, disconnected using ifconfig[7] command.
- Although it is easy to use epairs system, netgraph system has a great advantage over epair. Using pipe, network parameters can be changed and a more realistic virtual network can be created. Epair does not provide such functionality.

The epair is a pair of Ethernet-like software interfaces, which are connected back-to-back with a virtual cross-over cable. In order to connect two devices directly each epair ends can be moved to different devices. Listing 3.5 represent corresponding commands to achieve two directly connected devices. Assume previously two devices named n0 and n1 are created using jail commands mentioned in section 3.1.1.

```
1  root# ifconfig epair create
2  epair0a
3  root# ifconfig epair0a vnet n0
4  root# ifconfig epair0b vnet n1
5  root# jexec n0 ifconfig
6  lo0: flags=8008<LOOPBACK,MULTICAST> metric 0 mtu 16384
7          options=3<RXCSUM,TXCSUM>
8  epair0a: flags=8842<BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
9          ether 02:da:6a:00:04:0a
10 root# jexec n1 ifconfig
11 lo0: flags=8008<LOOPBACK,MULTICAST> metric 0 mtu 16384
12         options=3<RXCSUM,TXCSUM>
13 epair0b: flags=8842<BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
14         ether 02:00:00:00:05:0b
```

Listing 3.5: Connecting two devices with epair

First line of listing 3.5 creates two epair interfaces named epair0a and epair0b. Lines 2-3 move each interface to a virtual node. At the end, on line 4-5, it can be seen that virtual devices get epair interfaces.

In listing 3.6 four virtual devices are connected each other by a bridge device. Assume previously four devices (n0, n1, n2, n3) are created using Jail commands mentioned in section 3.1.1.

```
1  root# ifconfig epair create
2  epair0a
3  root# ifconfig epair create
4  epair1a
5  root# ifconfig epair create
6  epair2a
7  root# ifconfig epair create
8  epair3a
9  root# ifconfig epair0b vnet n0
10 root# ifconfig epair1b vnet n1
11 root# ifconfig epair2b vnet n2
12 root# ifconfig epair3b vnet n3
13 root# ifconfig bridge create
14 bridge0
15 root# ifconfig bridge0 addm epair0a
16 root# ifconfig bridge0 addm epair1a
17 root# ifconfig bridge0 addm epair2a
18 root# ifconfig bridge0 addm epair3a
```

Listing 3.6: Connecting four devices with bridge

Lines 1-3-5-7 create four pairs have the form epairXa-epairXb where X is the pair index(i.e first pair's index is 0, second's is 1 and so on). Lines 9-12 moves "b" ends to suitable virtual nodes. At line 13 a bridge object is created. Lines 14-18 add "a" sides of epairs are to bridge object. Bridge object acts as a switch and connects nodes.

In the earlier versions of Kovan, epair system is used for all linking operations. On the other hand, netgraph system is used because of its link parameter modification support in new versions. Kovan utilizes link parameter modification to look like a more realistic network. But, even -almost- all links are netgraph based, epair links are used for two exception cases. First, assume a bridge connection created as in figure 3.4 with netgraph but -for some reason- you want to give IP to "A" side of the connection. It is not possible to do such a configuration with netgraph but possible with epair. In Kovan, the same situation occurred where we want to connect whole virtual system to BaseSystem network and epair system is used for linking.

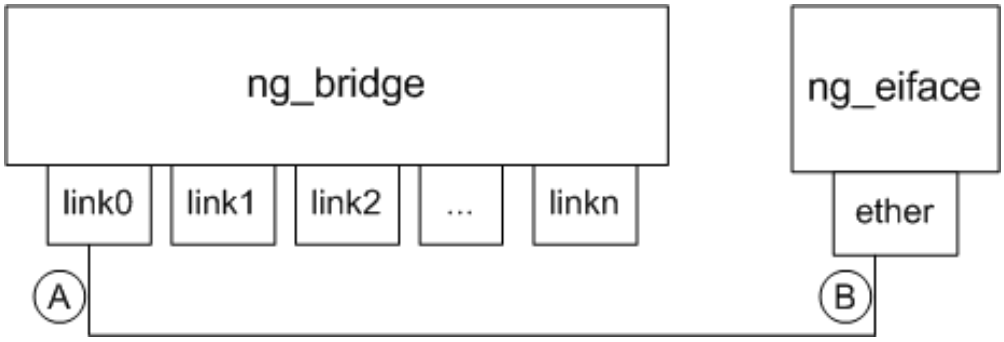

Figure 3.4: Kovan epair usage exception

The second case where epair system is used instead of epair is connecting QEMU/KVM images to virtual network. In order to connect these images bridge interface must be accessible via ifconfig system call. Therefore epair system is used in this advanced case, details are explained in section 8.1.

After linking nodes, Kovan has to give suitable IP addresses to interfaces and setup routing. Routing process is mentioned in section 3.1.4

3.1.4 Virtual Connectivity: Dynamic Routing

Kovan uses Quagga Routing suite as routing software. For IPv4 routing, distance vector routing protocol RIPv2 and for IPv6 routing Ripng is used. RIP was chosen for ease of configuration and small cpu footprint compared to link stat routing protocols. For each router, rip, ripng and zebra configuration files are generated by Kovan during start up phase. Configuration files are generated under router jail. The path is /usr/local/etc/quaga, following the name of the router. For example, if the router name is anarouter the configuration files:

```
/usr/local/etc/quagga/anarouter/ripd.conf
/usr/local/etc/quagga/anarouter/ripng.conf
/usr/local/etc/quagga/anarouter/zebra.conf
```

are generated. Quagga provides management through telnet management interface. Thus, the port numbers in Table 3.1 can be used for configuration of routing processes directly by using telnet.

3.2 Stateful Architecture

Kovan can be started with different configuration files. Each configuration can define different network topologies with different services. Kovan stores network flow, network statistics, bandwidth usages ...etc

Zebra	2601
Rip	2602
Ripng	2603

Table 3.1: Quagga Management Port Numbers

during execution. If user wants to start Kovan with another configuration current configuration's state should be saved. All "state carrying" files are archieved and moved to kovan_path/states/ path. State carrying files can be categorized in five topics:

- Kovan configuration files
- MRTG configuration files and produced web files
- NFSEN configuration files and database files
- NAGIOS configuration files and database files
- MYSQL database files
- Statistic graph files

kovanState's arguments is shown below. When saving a state user has to give a state name. kovanState adds a date string to the given name. When loading a state, target state name should be given to kovanState script.

```
root# ./kovanState
usage: kovanState --save state_name
       kovanState --load state_name
```

There is a list of all state carrying files below. These files are copied when save state command is called a tar archieve is created with given state name. If kovanState is called with loading mode, given state is unarchieved and files are moved to their paths. If a new state is loaded without saving current state, current state files are overwriten and state is lost.

```
# Kovan configuration files
nodeJailRoot/usr/local/kovan/etc

# MRTG configuration files and produced web files
monitorJailRoot/usr/local/etc/mrtg
monitorJailRoot/usr/local/www/mrtg

# NFSEN configuration files and database files
monitorJailRoot/usr/local/var/nfsen/profiles/live/*
monitorJailRoot/usr/local/var/nfsen/profiles-data/live/*
monitorJailRoot/usr/local/var/nfsen/profiles-stat/live/*.png
monitorJailRoot/usr/local/var/nfsen/profiles-stat/live/*.rrd
monitorJailRoot/usr/local/var/nfsen/run/*
monitorJailRoot/usr/local/etc/nfsen.conf

# NAGIOS configuration files and database files
monitorJailRoot/usr/local/etc/nagios/*.cfg
monitorJailRoot/var/spool/nagios/archives/*
monitorJailRoot/var/spool/nagios/checkresults/*
```

```
monitorJailRoot/var/spool/nagios/rw/*
monitorJailRoot/var/spool/nagios/status.dat
monitorJailRoot/var/spool/nagios/nagios.log
monitorJailRoot/var/spool/nagios/objects.cache

# MYSQL database files
monitorJailRoot/var/db/mysql/*

# Statistic graph files
monitorJailRoot/usr/local/www/stats/*.png
```

3.3 Monitoring

3.3.1 SNMP

Kovan uses Net-SNMP package for SNMP monitoring. Net-SNMP package is installed during jail setup by jailCreate.sh script. Sample snmpd.conf file is also copied from Kovan source code directory to router jail images /etc directory during installation. MRTG is used for drawing the bandwidth usage using SNMP. MRTG on the monitor host is configured by kovanMRTG script. For more information please refer to section 4.1. When reading the configuration file Kovan check the device type, if the device type is a router snmpd will launch and it bind to all ipv6 interfaces on the router. Although snmpd can listen both IPv4 and IPv6 requests, Kovan launch snmpd with an option to listen only IPv6 traffic on udp port 161. Snmpd is not bind to any IPv4 interface. For each router snmpd is started with the following arguments

```
/usr/local/sbin/snmpd -c /etc/snmpd.conf udp6:161
```

3.3.2 NETFLOW

Kovan use softflowd as flow generator and nfsen as an flow collector. Both programs are installed by jailCreate.sh script

Softflowd

Kovan launch softflowd if the virtual device type is a router. Related with netflow monitoring, Kovan look for the following parameters when parsing configuration file

```
if device type is a router
get router jail id
get number of interfaces on the router
if device type is a monitor
get monitor IPv6 address
get nfsen_start_port variable
```

Kovan use this information to launch a separate softflowd process for all interfaces on all routers. Since the flow collector Nfsen is installed on the Monitor host jail, IPv6 address of Monitor host is used as a destination. During kovan startup file processing nfsen_start_port is the first routers first interface. For the second interface the port number is incremented. When the (Buraya algoritma daha guzel bir sekilde if li yazilaXXXXX For each router softflowd is launched with the following arguments.

```
/usr/local/sbin/softflowd -v 9 -i interface -m 1000 -n[MonitorIPv6address]:port number
```

To further understand the netflow monitoring on Kovan, let us give an example. If we have:

- Two router, router1 and router2.

- Router1 has two interface and Router2 has the three interface
- Monitor IPv6 address 2001:a98:13:3000::12
- Nfsen_start_port is 9000

Kovan launch the following commands in Router1

```
/usr/sbin/softflowd -v9 -i eth0 -m 1000 -n[2001:a98:13:3000::12]9000
/usr/sbin/softflowd -v9 -i eth1 -m 1000 -n[2001:a98:13:3000::12]9001
```

and in Router2

```
/usr/sbin/softflowd -v9 -i eth0 -m 1000 -n[2001:a98:13:3000::12]9002
/usr/sbin/softflowd -v9 -i eth1 -m 1000 -n[2001:a98:13:3000::12]9003
/usr/sbin/softflowd -v9 -i eth2 -m 1000 -n[2001:a98:13:3000::12]9004
```

NFSEN

Kovan use kovanNfsen to install, configure and run nfsen program. After running Kovan, kovanNfsen should be launch to configure nfsen.conf file in monitor jail. KovanNfsen use the same port generation algorithm as Kovan thus flow collector and generator are in sync with respect to router port tuple. KovanNfsen install the nfsen web pages on http://monitor ipv6 address/nfsen. Please refer section 4.3 for more information.

3.3.3 Service Monitoring

Nagios is used for monitoring Kovan framework. Nagios not only checks the up/down status of created routers and nodes but also running processes on this hosts automatically. In short,for each router node nagios check for the:

- system up/down status
- snmpd process
- softflowd process
- zebra process
- ripnd process
- rip process

For each node:

- System up/down status
- Kovan services (i.e http, smtp, ftp, dns)

are also monitored by nagios.

Bibliography

[1] Argos, an emulator for capturing zero-day attacks, http://www.few.vu.nl/argos/

[2] Nephentes, a tool to collect malware, http://www.honeynet.org/project/nepenthes

[3] http://www.freebsd.org/doc/handbook/jails.html

[4]

[5]

[6]

[7]

Chapter 4

Tools

Kovan has many scripts for configuring the tools explained in "'section 3.3"' . The monitor jail which is created during kovan installation sets up these tools. Kovan user can configure the monitor jail and all of its tools manually but this takes too much time. There are several scripts in kovan/bin directory which are written to make configuration process easier and faster. All scripts have to get the kovan configuration file as a parameter to parse kovan global configuration. Also they have the same functions which are install, configure and run.

- install : checks for necessary configuration files, creates configuration directories and copies required files into these directories. If a configuration file already exists, it overwrites the existing file with the newly created one.
- configure : prepares the configuration files by executing regular expressions.
- run : starts the service and if the service running then restarts it.

Use "'install"' scripts only to begin a new configuration of the tool. Please note that, if you make some changes on configuration files manually and then if you execute "'install"' for this tool, it will overwrites the configuration files and you lost your changes. If you already have your tool configured and you want to update your tool according to the new configuration use "'configure"'. For example nagios is already configured and you've added a new host to the system and you want nagios to begin monitoring this host, please type :

```
root#./kovanNagios -f kovan_conf_file --configure
```

You can use these three parameters together in one command :

```
root#./kovanNagios -f kovan_conf_file --install --configure --run --mail mail@mail.com
```

Order of the parameters is not important. For each parameter you should give additional parameters like above (–mail). Also you can see the usage of scripts by typing –help :

```
root#./kovanNagios --help
Usage:  ./kovanNagios [-c kovan_conf_file] [--mail mail_address] [--snmp snmp_community]
                      [--install|--configure|--execute]
Options :
        -c kovan_conf_file                : kovan configuration file
        --snmp snmp_community             : snmp community name
```

```
        --mail mail_address                : mail address of nagios user
        --install                          : checks for files and copy them for configuring
        --configure                        : executes regex on files and configures them
        --execute                          : runs the binary
```

4.1 kovanNagios

Nagios is an open source monitoring tool that has many monitoring capabilities. The main usage of nagios in kovan is checking routers,nodes and all services if they're up or not. 'kovanNagios' script is written for configuring nagios. It prepares and configures files, finally runs the nagios binary. kovanNagiosCheck plugin is written for nagios to check ipv6 devices with ping6. It can also be used to check running processes on routers such as softflowd, zebra etc. To see the usage of kovanNagios script type :

```
root#./kovanNagios --help
 USAGE:   kovanNagios  -f kovan_conf_file  [--install|--configure|--run] --mail mail_address
 --community snmp_community
         kovan_conf_file    : kovan configuration file
         snmp_community      : snmp community name
         mail_address        : mail address of nagios user
         install             : checks for files and copy them for configuring
         configure           : executes regex on files and configures them
         run                 : runs the binary
```

Nagios has a main configuration file nagios.cfg and other configuration files which are resource.cfg, cgi.cfg, commands.cfg, contacts.cfg, templates.cfg , timeperiods.cfg and localhost.cfg. When nagios is installed for the first time, these files has *.cfg-sample extension by default. To prepare these files to configuration, they has to be copied as *.cfg extension in nagios folder. To make this type :

```
root# ./kovanNagios -f kovanTopology.conf --install
```

For running nagios web interface correctly, htttpd.conf also has to be configured. 'install' checks for httpd.conf file if it exists. Install checks for kovanNagiosCheck plugin which is installed by the time monitor jail is installed. If any of the configuration files can not be found, program exit with 'install is not executed' error. To configure nagios files and httpd.conf file type : root ./kovanNagios -f kovanTopology.conf –mail admin@administrator.com –community public –configure httpd.conf completed kovanNagiosCheck, check-kovan-alive,check$_u dpcommandsdefinedcommands.cfgcompletedcheck-kovan-aliveregexoktemplates.cfgcompletedcontac$
1 For configuration, all scripts have to parse kovanTopology.conf to get host and service informations. 'kovanNagios' gets 2 additional parameters. First, 'kovanNagiosCheck' plugin needs for snmp community name which is given with –community parameter, to query devices with snmpwalk and mail address parameter is needed by contacts.cfg file to send alerts of devices to kovan administrator mail address. The main task of configuration is mostly done with regular expressions. For example web directory paths and cgi configurations of nagios are added to httpd.conf file as above :

```
##kvn_nagios_s
    <Directory /usr/local/www/nagios>
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all

    </Directory>
```

```
    <Directory /usr/local/www/nagios/cgi-bin>
    AllowOverride None
    Options ExecCGI
    Order allow,deny
    Allow from all

    </Directory>

    ScriptAlias /nagios/cgi-bin/ /usr/local/www/nagios/cgi-bin/
    Alias /nagios /usr/local/www/nagios
##kvn_nagios_f
```

All scripts in kovan has a commenting standart which is

```
##kovan_scriptname_s
...
...  regex tasks
...
##kovan_scriptname_f
```

When any part of a configuration file is edited by a script, that parts are signed with this strings. If configure is executed more than one, kovanNagios checks for this strings and edits between them according to the statement. After httpd.conf is configured, kovanNagios :

- defines kovanNagiosCheck, check-kovan-alive and check_udp commands in commands.cfg file,
- changes check_command of freebsd-servers check-host-alive to check-kovan-alive in templates.cfg,
- defines mail address of nagios admin user in contacts.cfg,
- configures host and service definitions in localhost.cfg,
- changes authentication of web interface to none in cgi.cfg

If any error happens during configuration, program exists with 'configure is not executed' message. Finally to run nagios type :

```
root# ./kovanNagios --run
Performing sanity check of nagios configuration: OK
Starting nagios.
run is executed : 1
```

4.2 kovanMrtg

As mentioned in previous sections, MRTG is used for monitoring bandwidth usage in the virtual network. MRTG has a script named 'configm4aker' that creates mrtg configuration file. Also to create web files from this configuration file mrtg has indexmaker script. Mrtg runs by crontab in every five minutes to update usage statistics in web interface. To see the usage of kovanMrtg script type :

```
root# ./kovanMrtg --help
 USAGE:   kovanMrtg   -f kovan_conf_file  [--install|--configure|--run] --community snmp_community
         kovan_conf_file     : kovan configuration file
         snmp_community      : snmp community name
         install             : checks for files and copy them for configuring
         configure           : executes regex on files and configures them
         run                 : runs the binary
```

To start mrtg configuration first type :

```
root# ./kovanMrtg -f kovanTopology.conf --install
```

After install is executed, mrtg configuration and web directories are created. Also crontab file can be seen under /tmp directory. To configure mrtg files type :

```
root# ./kovanMrtg -f ../etc/kovanTopology.conf --community public --configure
```

During configure, kovanMrtg :

- uses configmaker script to create mrtg.conf file with giving snmp community name and router ip parameters parsed from kovanTopology.conf.
- executes indexmaker script to create web files in mrtg web directory.
- finally adds cron string to crontab file and finishes configuration.

```
5,10,15,20,25,30,35,40,45,50,55  *  *  *  *  /usr/local/bin/mrtg /usr/local/kovan/mrtg/mrtg.conf
```

Finally to run nagios type :

```
root# ./kovanMrtg --run
crontab is executed
run is finished
```

Mrtg web files are updated in every five minutes after crontab is run.

4.3 kovanNfsen

NfSen generates stats from collected NetFlow data from routers in the virtual network and displays through a web interface. To see the usage of kovanMrtg script type :

```
#root ./kovanNfsen --help
 USAGE:  nfsenCreate -f kovan_conf_file [--install|--configure|--run] [-n nfsen-dist.file]
                        -s subdirectory_value -u nfsen_user --www-user web_user --www-group web_group
         conf_file             : kovan configuration file
         nfsen-dist.file       : nfsen-dist.file path
         subdirectory_value    : subdirectory value, between 1 <= s <= 9
         nfsen_user            : name of the nfsen processes user
         web_user              : name of the user that runs web process
         web_group             : name of the group that runs web process
         install               : checks for files and copy them for configuring
         configure             : executes regex on files and configures them
         run                   : runs the binary
```

To prepare nfsen and apache configuration files type :

```
root# ./kovanNfsen -f kovanTopology.conf --install
```

To configure httpd.conf and nfsen configuration files type :

```
root# ./kovanNfsen -f kovanTopology.conf --configure -s 4
sources is changed.
subdirectory is changed.
nfsen.conf is ready
httpd.conf is ready
configure is executed : 1
```

During configuration, kovanNfsen :

- gets routers interface lists from kovanTopology.conf file,
- matches nfsen collecting ports according to routers flow generator ports,
- assigns a random color for every router in web interface,
- changes nfsen.conf with the optional parameters given for configuration (like subdirectory value 4 in this example),
- finally adds nagios web directory path to httpd.conf.

Finally to run nagios type :

```
root# ./kovanNfsen --run
Reconfig: No changes found!
Starting nfcapd: comur1_eth0[84837] anarouter_eth2[84840] anarouter_eth1[84843] ituh_eth0[84846] comur1
run is finished
```

4.4 kovanApachePhp

All monitoring tools of kovan use apache http server for serving web interfaces. Apache is installed with php support during monitor jail setup. kovanApachePhp is written for configuring php.ini,httpd.conf and /etc/hosts files. To see the usage of kovanApachePhp script type :

```
root# ./kovanApachePhp --help
 USAGE:  kovanApachePhp -f kovan_conf_file [--install|--configure|--run]
         kovan_conf_file       : kovan configuration file
         install               : checks for files and copy them for configuring
         configure             : executes regex on files and configures them
         run                   : runs the binary
```

To prepare configuration files type :

```
root# ./kovanApachePhp -f kovanTopology.conf --install
/usr/local/etc/apache22/httpd.conf does not exists. Please find and copy an httpd.conf file to /usr/loc
install is not finished
```

By default apache puts an untouched httpd.conf to apache directory, and php puts sample php-ini files to /usr/local/etc. kovanApachePhp checks for configuration files, if httpd.conf does not exists, script exits with an error as seen below. After httpd.conf file is put to /usr/local/etc/apache22, kovanApachePhp copies php.ini file and gives :

```
/usr/local/etc/apache22/httpd.conf is ok.
/usr/local/etc/php.ini is ok.
install is executed
```

Afer that, to start configuration type :

```
root# ./kovanApachePhp -f kovanTopology.conf --configure
/usr/local/kovan/monitor/usr/local/etc/apache22/httpd.conf    ok.
httpd regex completed.
httpd.conf is ready.
hosts regex completed.
hosts is ready.
php.ini regex completed.
php.ini is ready.
configure finished
```

During configuration, kovanApachePhp :

- configures apache for binding to ipv6 address,
- changes directives of httpd.conf for php module support,
- adds monitor jail ipv6 address and hostname to /etc/hosts file for apache,
- defines timezone value in php.ini file.

To run apache server type :

```
root# ./kovanApachePhp --run
Performing sanity check on apache22 configuration:
Syntax OK
Starting apache22.
```

4.5 kovangraph

kovan_graph draws attack statistics graphics from information taken from services and blackhole. All service attack logs and blackhole flows are send to monitor device periodically (section 14). kovan_graph script is called every hour for drawing statistics of previous hour and every day for previous day from cron.

kovan_graph takes 5 arguments as shown below.

```
root# ./kovan_graph
usage: kovan_graph --outputfile NAME --xaxis TITLE --yaxis TITLE --query QUERY --header HEADER
```

First argument (–outputfile) determines the name of the output graph file. X and Y axis values are axis names of the graph. Fourth argument (–query) is the sql query used for retreiving suitable rows and coloumns from mysql database. It is important to note that, sql query must return a two coloumn result; first column will be used in x axis and the second column will be used in y axis. Last argument is the header of the graph. Kovan, provides sh scripts for calling kovan_graph scripts.

4.6 kovanManageLinks

kovanManageLinks is a tool for modifying link properties. Using kovanManageLinks, all links properties (except first link connecting to physical interface) can set, reset and changed. Below command shows the arguments of the script:

```
roo# ./kovanManageLinks
usage: kovanManageLinks --all [--show]
       kovanManageLinks --show link_name
       kovanManageLinks --reset link_name
       kovanManageLinks --delay=X(s|ms|us)
            --bandwidth=Y(gbps|mbps|kbps|bps)
            --upstream_ber=Z --downstream_ber=T link_name
```

“–all” flag lists all current modifiable links. To give an example, we can use kovanManageLinks over a working topology show in figure 4.1.

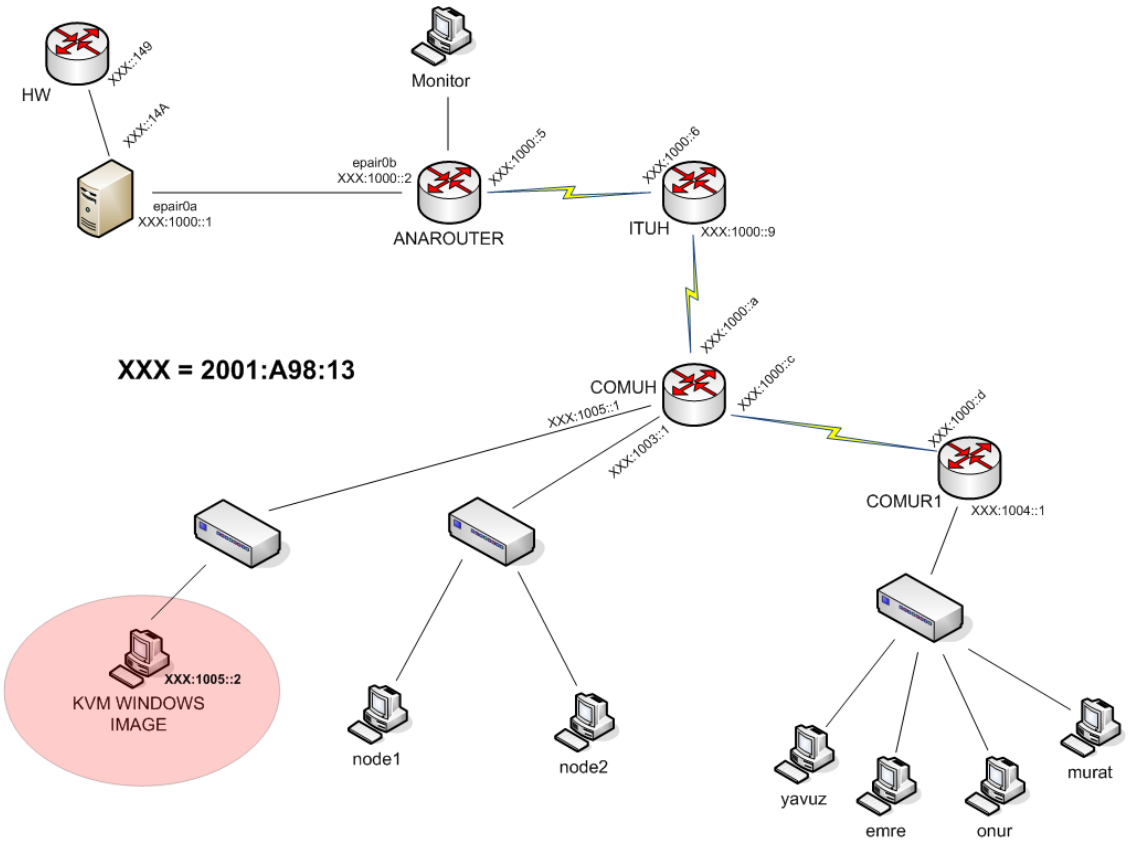

Figure 4.1: Example Kovan topology

The result of –all is shown in listing 4.1.

```
1  root# ./perl kovanManageLinks --all
2  ituh-comuh
3  yavuz-bridge2
4  onur-bridge2
5  cdep1-bridge1
6  cdep2-bridge1
7  comuh-comur1
8  comur1-bridge2
9  anarouter-monitor_host
10 anarouter-ituh
11 comuh-bridge1
12 murat-bridge2
13 emre-bridge2
```

Listing 4.1: Listing all kovan links

"–show" argument is used for printing detailed link(s). Any link listed in listing 4.1 can be detailly demonstrated as in 4.3:

```
1  root# ./kovanManageLinks --show comuh-comur1
2  comuh-comur1:
```

```
3       Bandwidth: default
4       Delay: default
5       UpStream BER: default
6       DownStream BER: default
```

Listing 4.2: Details of one link

In addition to detailing a single link, kovanManageLinks can be used to list all links details as shown in **??**

```
1  root# ./kovanManageLinks --show --all
2  ituh-comuh:
3      Bandwidth: default
4      Delay: default
5      UpStream BER: default
6      DownStream BER: default
7
8  yavuz-bridge2:
9      Bandwidth: default
10     Delay: default
11     UpStream BER: default
12     DownStream BER: default
13
14 onur-bridge2:
15     Bandwidth: default
16     Delay: default
17     UpStream BER: default
18     DownStream BER: default
19
20 cdep1-bridge1:
21     Bandwidth: default
22     Delay: default
23     UpStream BER: default
24     DownStream BER: default
25
26 cdep2-bridge1:
27     Bandwidth: default
28     Delay: default
29     UpStream BER: default
30     DownStream BER: default
31
32 comuh-comur1:
33     Bandwidth: default
34     Delay: default
35     UpStream BER: default
36     DownStream BER: default
37
38 comur1-bridge2:
39     Bandwidth: default
40     Delay: default
41     UpStream BER: default
42     DownStream BER: default
43
```

```
anarouter-monitor_host:
    Bandwidth: default
    Delay: default
    UpStream BER: default
    DownStream BER: default

anarouter-ituh:
    Bandwidth: default
    Delay: default
    UpStream BER: default
    DownStream BER: default

comuh-bridge1:
    Bandwidth: default
    Delay: default
    UpStream BER: default
    DownStream BER: default

murat-bridge2:
    Bandwidth: default
    Delay: default
    UpStream BER: default
    DownStream BER: default

emre-bridge2:
    Bandwidth: default
    Delay: default
    UpStream BER: default
    DownStream BER: default
```

Listing 4.3: Details of one link

kovanManageLinks script is used not only for listing links but also to modify them. Link properties such as delay, bandwidth, upstream BER (bit error rate) and downstream BER can be set via kovanManageLinks script. Delay property can be set with argument "–delay=X(s—ms—us)" where X is the magnitude and "s—ms—us" are possible delay units. Bandwidth property can be set with argument "–bandwidth=Y(gbps—mbps—kbps—bps)" where Y is the magnitude and "gbps—mbps—kbps—bps" are possible bandwidth units. BER values are given as integer, if given BER value is T than bit error rate is set to 1/T.

Listing 4.4 shows an example about changing link properties. In this example, anarouter-ituh link's delay is set to 3 millisecond and bandwidth is set to 5 megabits per second.

```
[root@ ~/KOVAN/kovan_host/bin]# perl kovanManageLinks --delay=3ms --bandwidth
    =5mbps anarouter-ituh
anarouter-ituh:
    Bandwidth: 5Mbps
    Delay: 3ms
    UpStream BER: default
    DownStream BER: default
```

Listing 4.4: Modifying one link

4.7 kovanMonitorServices

As explained above sections, all scripts (except kovanManageLinks) run with the same parameters : "install", "configure" and "execute". kovanMonitorServices gets all parameters and runs all scripts externally and completes the configuration of Monitor jail services.

kovanMonitorServices usage is :

```
root# ./kovanMonitorServices
usage:  kovanMonitorServices --conf conffile --snmp snmppasswd --mail adminmail
                             [--list] [--install|--configure|--execute] [--all]
```

Additionally, "–list" parameter is used with one of the "–install, –configure, –execute" or "–all" parameters. The script can be used as other scripts with one of the parameters or using"–all" parameter which runs install, configure and execute in order.

Chapter 5

Getting Started

5.1 Advice For Beginners

- If you are novice FreeBSD user we recommend using pre configured Kovan KVM/QEMU images.
- If you are intermediate FreeBSD user, it is reccomended to follow 5.3 Quick Installation Guide.
- If you are an expert of FreeBSD, you can choose the hard way by following the 5.4 step by step installation guide.
- The installation scripts should work seamlessly with FreeBSD 8.1. Therefore, it is recommended to use installation script.

5.2 Must-have Configurations

- You should install Kovan to freshly installed FreeBSD system. Installing Kovan on a production system will probably fail and not recommended.
- IP and DNS setup should be completed before installing Kovan. If Kovan is used in a pure IPv6 network, DNS server configuration in /etc/resolv.conf file should be arranged accordingly. Likewise, if Kovan is used in dual stack environment, DNS servers should be available for both protocols.
- Kovan installation scripts use bash shell. Bash should be installed before running any Kovan scripts.
- Kovan installation scripts is run with the set -e option. If there is any error during installation, setup will exit.
- Kovan install scripts use FreeBSD Ports system without FORCE_PKG_REGISTER setting. If the package is installed before the startup system will fail. If this is the case set the FORCE_PKG_REGISTER to 1 (i.e export FORCE_PKG_REGISTER=1)

5.3 Quickstart installation guide

Quick installation requires two steps and only two scripts are need to run. For the first step, preinstall.sh script should be run without any argument. preinstall.sh performs the following actions:

- Updating FreeBSD ports and sources tree.
- Compiling FreeBSD kernel with VIMAGE support

- Setting up IPv4/IPv6 forwarding
- Setting up IPC parameters
- Installing perl, p5 -YAML -Tiny, p5 Getopt Long, automake,snort ports

After finishing these steps preinstall.sh reboots the system. Please note that preinstall.sh should be run after IP and DNS setup with the root privileges. The first step of installation may take approximately 1 hour. The time depends on the hardware configuration of the connection speed.

The second step install.sh script is used. The script is used to

- Setup monitor, router, node and blackhole jails
- Compile kovan source codes
- Install the compiled binary packages

Install.sh script requires 3 arguments.

1. PREFIX Target Path that Kovan will be installed (i.e /usr/local/kovan
2. IP A secondary IP address which are in the same subnet with the primary IP address. This IP address is used for connecting jails to internet
3. INTERFACE The name of the physical interface that will be used to connect the internet.

For example, if your physical interface is em0 with the IP address 10.10.20.4 and if you want to install kovan to /usr/local/kovan folder the command is

```
root# ./install.sh /usr/local/kovan 10.10.20.4 em0
```

The second step of installation may take approximately 2 hours .The time depends on the hardware configuration and the connection speed.

5.4 Step-by-step installation guide

5.4.1 Prepare FreeBSD for Kovan installation

Update FreeBSD source tree (Optional)

Cvsup port can be used for updating FreeBSD ports tree.

```
root#pkg_add -r cvsup-without-gui
```

Create a file (/etc/stable-supfile) containing the following lines. Do not forget to change the RELENG tag according to your FreeBSD release

```
        *default host=tulumba.ulakbim.gov.tr
        *default base=/usr
        *default prefix=/usr
        *default release=cvs tag=RELENG_8_1
        *default delete use-rel-suffix
        src-all
```

The source tree can be updated by running cvsup command

```
root#cvsup  /stable/supfile
```

Cron can be used for automating this process

```
30 12 * * * /usr/sbin/portsnap update
```

Update FreeBSD ports tree (Optional)

Use portsnap command to FreeBSD ports tree

```
root#portsnap -s portsnap2.freebsd.org fetch
root#portsnap extract
```

The other alternative is using cvsup for updating FreeBSD ports tree. The following stable-supfile updates both source and ports tree.

```
*default host=cvsup.freebsd.org
*default base=/usr
*default prefix=/usr
*default release=cvs tag=RELENG_8_1
*default delete use-rel-suffix
src-all
ports-all tag=.
```

Compile Kernel with VIMAGE support

For FreeBSD 8.1 and above add the following line to KERNEL configuration file (i.e /usr/local/sys/amd64/conf/GENERIC)

```
options VIMAGE
```

If you are using FreeBSD 8.0 you should also comment the SCTP lines

```
#options SCP
```

Compile the kernel (assuming your kernel configuration file /usr/local/sys/amd64/conf/GENERIC)

```
root#cd /usr/src/sys/amd64/conf
root#config GENERIC
root#cd ../compile/GENERIC/
root#make cleandepend && make depend && make && make install
```

Reboot the system to take settings effect.

Enable IPv6 and IPv6 routing

Add following lines to /etc/rc.conf to enable IPv6 on FreeBSD

```
ipv6_enable="YES"
```

Add following lines to /etc/sysctl.conf

```
net.inet.ip.forwarding=1
net.inet6.ip6.forwarding=1
security.jail.allow_raw_sockets=1
```

Setup IPC settings

Add the following lines to /boot/loader.conf file.

```
kern.ipc.shmall=32768
kern.ipc.shmmax=134217728
kern.ipc.semmap=256
kern.ipc.semmni=256
kern.ipc.semmns=512
kern.ipc.semmnu=256
```

5.4.2 Installing Additional Ports

Kovan use ezjail port for setup jails, to install this port type

```
root#cd /usr/ports/sysutils/ezjail
root#make -DBATCH install clean
root#ezjail-admin install
```

Setup the following ports as well

```
root# cd /usr/ports/lang/perl5.10 && make -DBATCH install clean
root# cd /usr/ports/textproc/p5-YAML-Tiny/ &&   make -DBATCH    install clean
root# cd /usr/ports/devel/p5-Getopt-Long &&   make -DBATCH  install clean
root# export CONFIGURE_ARGS="--enable-ipv6 --enable-gre --enable-targetbased
--enable-decoder-preprocessor-rules --enable-zlib"
root# cd /usr/ports/devel/automake111 && make -DBATCH install clean
root# cd /usr/ports/security/snort && make -DBATCH -DWITH_SNORTSAM install clean
```

5.4.3 Jail Setup

There are four types of Kovan jail, router, monitor, node and blackhole. All of them should be installed. For each type, different set of programs are need to compiled using FreeBSD ports tree. Although it takes much more time to setup, compiling option is preferred to precompiled packages intentionally to make sure that the ports have an IPv6 support. To setup jails yo need to:

1. Use ezjail port (i.e ezjail-admin create)
2. Install jails under the Kovan install prefix (i.e /usr/local/kovan)

3. Create a directory hierarcy with respect to jail names (i.e /usr/local/kovan/montitor for monitor jail)

4. /usr/ports should be mount using mount_nullfs command (i.e mount_nullfs -o ro /usr/ports /usr/jail_path/usr/ports)

5. Run the following commands for each type of jail

Node

```
root# pkg_add -r bash
root# mkdir -p /usr/local/kovan/etc
root# cd /usr/ports/ports-mgmt/portaudit && make install clean
root# /usr/local/sbin/portaudit -Fda
root# cd /usr/ports/textproc/p5-YAML-Tiny/ ; make -DBATCH -DFORCE_PKG_REGISTER install clean
root# cd /usr/ports/devel/p5-Getopt-Long ; make -DBATCH -DFORCE_PKG_REGISTER install clean
```

Router

```
root# pkg_add -r bash
root# mkdir -p /usr/local/kovan/etc
root# cd /usr/ports/ports-mgmt/portaudit && make install clean
root# /usr/local/sbin/portaudit -Fda
root# cd /usr/ports/net-mgmt/net-snmp ; make -DBATCH -DWITH_IPV6 install clean
root# cd /usr/ports/net/quagga && make  -DBATCH -DWITH_RTADV -DWITH_SNMP install clean
root# cd /usr/ports/textproc/p5-YAML-Tiny/; make -DBATCH    -DFORCE_PKG_REGISTER install clean
root# cd /usr/ports/devel/p5-Getopt-Long; make -DBATCH  -DFORCE_PKG_REGISTER install clean
root# cd  /usr/ports/net-mgmt/softflowd ; make -DBATCH install clean
```

Monitor

```
root# cd /usr/ports/ports-mgmt/portaudit && make install clean
root# /usr/local/sbin/portaudit -Fda
root# pkg_add -r bash
root# mkdir -p /usr/local/kovan/etc
root# cd /usr/ports/www/apache22 && make  -DBATCH install clean
root# cd /usr/ports/lang/php5 && make -DBATCH -DWITH_APACHE -DWITH_IPV6
root# cd /usr/ports/net-mgmt/net-snmp && make -DBATCH -DWITH_IPV6 install
root# cd /usr/ports/net-mgmt/nfsen && make  -DBATCH install clean
root# cd /usr/ports/net-mgmt/mrtg/&& make -DBATCH  -DWITH_IPV6 -DWITH_SNMP install clean
root# cd /usr/ports/net-mgmt/nagios-plugins && make -DBATCH -DWITH_JAIL -DWITH_IPV6
-DWITH_NETSNMP -DWITH_FPING install clean
root# /usr/sbin/pw groupadd nagios
root# /usr/sbin/pw adduser nagios nagios
root# cd /usr/ports/net-mgmt/nagios  && make -DBATCH  install clean
root# cd /usr/ports/textproc/p5-YAML-Tiny/; make -DBATCH install clean
root# cd /usr/ports/devel/p5-Getopt-Long; make -DBATCH  install clean
root# cd /usr/ports/databases/mysql51-server ; make -DBATCH  install clean
root# cd /usr/ports/databases/libdbi-drivers ; make -DBATCH -DWITHOUT_PGSQL -DWITHOUT_SQLITE3
 -DWITH_MYSQL install clean
root# cd /usr/ports/sysutils/syslog-ng3 ; make -DBATCH -DWITH_SQL -DWITH_IPV6 -DWITH_PCRE
 -DWITHOUT_SPOOF install clean
root# /usr/local/etc/rc.d/mysql-server onestart
root# echo "create database kovan" | mysql -uroot
```

```
root# echo "CREATE USER 'logger'@'localhost' IDENTIFIED BY '231905'" | mysql -uroot
root# echo "GRANT ALL ON kovan.* TO 'logger'@'localhost'" | mysql -uroot
root# mysqladmin -u root password 231905
root# /usr/local/etc/rc.d/mysql-server onestop
root# cd /usr/ports/databases/p5-DBI ; make -DBATCH  install clean
root# cd /usr/ports/databases/p5-DBD-mysql ; make -DBATCH  install clean
root# cd /usr/ports/graphics/p5-GD-Graph ; make -DBATCH  install clean
```

Blackhole

```
root# pkg_add -r bash
root# mkdir -p /usr/local/kovan/etc
root# cd /usr/ports/ports-mgmt/portaudit && make install clean
root# /usr/local/sbin/portaudit -Fda
root# cd /usr/ports/textproc/p5-YAML-Tiny/; make -DBATCH -DFORCE_PKG_REGISTER install clean
root# cd /usr/ports/devel/p5-Getopt-Long; make -DBATCH  -DFORCE_PKG_REGISTER install clean
root# cd /usr/ports/net-mgmt/net-snmp && make -DBATCH -DWITH_IPV6 install clean
root# cd /usr/ports/net-mgmt/softflowd ; make -DBATCH install clean
```

5.4.4 Kovan installation

In the Kovan source directory type

```
root#cd kovan_host
root#make
root#make install
root#cd ..
root#cd kovan_jail
root#make
root#make install
```

that is all

5.5 Using pre-installed VM images

We only provide AMD64 qemu/kvm images for Kovan. To setup VM version of Kovan first download the KVM files from http://www.ipv6.net.tr/kovan. You need a a working KVM/QEMU setup for your base operating system. This section describes QEMU setup for FreeBSD, for Linux KVM please look for your favorite Linux distro documentation.

5.5.1 FreeBSD/QEMU Setup

Please see the recent version of QEMU FreeBSD setup on http://wiki.freebsd.org/qemu page: **Install**
Install the qemu port as

```
root#cd /usr/ports/emulators/qemu
root#make -DBATCH install clean
root#cd /usr/ports/emulators/kqemu-kmod
root#make -DBATCH install clean
```

Load the necessary kernel modules

```
root#kldload kqemu
root#kldload aio
```

Autoloading of kqemu (and aio) at boot can be enabled by adding the following line to your /etc/rc.conf.

```
kqemu_enable="YES"
```

Configure & Run

Create a brigde and a tap device and connect the tap interface to bridge (Assuming your physical ethernet device em0)

```
root#ifconfig bridge
root#ifconfig tap
root#ifconfig bridge0 addm em0 addm tap0 up
```

Start Kovan FreeBSD image with 2 Gbyte of memory 2 core on vnc 1

```
qemu-system-x86_64 -m 2048 -smp 2  -net nic -net tap,name=tap0, -hda kovan.amd64.kvm.1.0.img
-boot c -vnc :1
```

Please see qemu documentation for more information.

Chapter 6

Configuring Kovan

Kovan uses a single configuration file to create a virtual network topology and distribute fake/real services over virtual nodes. Kovan configuration file contains:

- Constant values such as jail paths, physical interface name, kovan base dir ...etc
- Virtual device definitions and device specific information such as default routes.
- Virtual network connections that connects virtual devices.
- Definitions of services that will be run on virtual devices.

YAML[1] is chosen as configuration file format because it is more human-readable than other common formats XML[2] and JSON[3]. Following sections describes the format of Kovan configuration file with examples.

6.1 Constants

There are two constant values that must be set for a proper execution of Kovan which are:

- **physical_ether**: The name of the ethernet interface that all virtual topology will be connected. Kovan's virtual network has to be connected to a real interface to access global internet.
- **kovan_dir**: The installation path of the Kovan application. The default installation path is "/usr/local/kovan" but it may be set to a different value during installation (see section 5.4.4).

Listing 6.1 is taken from example configuration file. In this example Kovan network is connected to re0 interface and installation path is /usr/local/kovan.

```
1 physical_ether: re0
2 kovan_dir: /usr/local/kovan
```

Listing 6.1: Kovan Configuration Constants Section

6.2 Devices

All network devices (i.e components) are defined in this section. There four types of devices (router, node, monitor, blackhole) that can be defined, according to device type some fields may be discarded.

- **name**: Determines the name of the virtual device and the name will be seen at **jls** output. Virtual device can be accessed via "name" using **jexec name cmd** command.

- **type**: Type field determines the jail template that will be used while creating virtual device. Possible values are: router, node, monitor and blackhole. Router and node types are obvious, use router type for virtual routers and node type for server/client computers. Monitor type is used for a special device, each configuration can has at most 1 device with monitor type. If a monitor device is added to network administrator can use this host to monitor network with bundle of tools (see section 3.3). Blackhole device is used for creating blackholes. In order to create blackholes, one or more unused subnets should be forwarded to blackhole device. Blackhole can use kovan_dump service to capture blackhole flow.

- **distribute**: This parameter is only applicable for nodes of type **router**. If set to value 1 this router will distributes its default routes.

- **default_route**: List default static routes of virtual device at this parameter. Default routes must have the form "-inet default 10.10.10.1" for IPv4 and "-inet6 default 2001:a98:13:4000::1" for IPv6.

Listing 6.2 is taken from example configuration file. In this example localhost, anarouter and ituh routers are defined. Distribute parameter is set for anarouter, anarouter distributes its default router. Router ituh does not have any route information it gets route information via dynamic routing. A monitor device named monitor_host is defined, this host will hold all monitoring software. Two user computers are defined with type node and they have default IPv4 routes.

```
1  devices:
2   - name: localhost
3     type: router
4     default_route:
5      - "-inet6 -net 2001:a98:13:4000::/60 2001:a98:13:4000::2"
6      - "-inet -net 10.10.10.0/24 10.10.10.2"
7   - name: anarouter
8     type: router
9     distribute: 1
10    default_route:
11     - "-inet6 default 2001:a98:13:4000::1"
12     - "-inet default 10.10.10.1"
13  - name: ituh
14    type: router
15    distribute: 0
16  - name: monitor_host
17    type: monitor
18    default_route:
19     - "-inet6 default 2001:a98:13:4000::11"
20     - "-inet default 10.10.30.1"
21    nfsen_start_port: 9000
22  - name: user1
23    type: node
24    default_route:
25     - "-inet default 10.10.60.1"
26  - name: user2
27    type: node
28    default_route:
29     - "-inet default 10.10.60.1"
```

Listing 6.2: Kovan Configuration Constants Section

6.3 Connections

Connection section defines virtual links between virtual devices. Administrator does not have to use all virtual devices in connections filed but each devices used in connection must be defined in devices field. There are two types of connections in Kovan: bridge and direct connection. Bridge connections is used for connecting more than two devices whereas direct connections connects only two device. Below are the parameters that may be used in connections filed.

- **type**: This parameter determines the connection type, possible types are bridge and direct. If the connection type is direct there must be two devices in peers parameter otherwise any number of devices can be listed in peers section.
- **bandwidth**: (only applicable for direct connections) This parameter determines the bandwidth value on the direct connection. Possible bandwidth units are: Gigabits Per Second(Gbps), Megabits Per Second(Mbps), Kilobits Per Second(Kbps) and Bits Per Second(bps).
- **delay**: (only applicable for direct connections) This parameter determines the delay value on the direct connection. Possible dela units are: Microsecond(us), Milisecond(ms) and Second(s).
- **downstream_ber**: (only applicable for direct connections) This parameter determines the bit error rate of downstream connection. This field excepts integer values. If "2" is given as BER value, downstream bit error rate is set to 1/2.
- **upstream_ber**: (only applicable for direct connections) This parameter determines the bit error rate of upstream connection. This field excepts integer values. If "2" is given as BER value, bit error rate is set to 1/2.
- **peers**: This is just a wrapper parameter, upstream below peers keyword virtual devices used in connection are listed.
  - **name**: The name of the virtual device defined in devices section.
  - **ip_addresses**: This field is only applicable for direct connections. Direct connections can be seen as a cable directly connects two devices. Naturally this cable has two ends and each end has a IPv4/IPv6 addresses. ip_addresses parameter determines the IP address of the corresponding end of the direct connection. It is not compulsory to use both IPv4 and IPv6 addresses unless a dual-stack network is designed.
  - **b_ip_address**: This field is only applicable for bridge connections. When connecting a device to a bridge connection has two ends namely bridge end and node end. This parameter determines the IP addresses of the bridge end of the bridge connection.
  - **n_ip_address**: This field is only applicable for bridge connections. When connecting a device to a bridge connection has two ends namely bridge end and node end. This parameter determines the IP addresses of the node end of the bridge connection. If node side IPv6 address is set "auto" node gets IPv6 address automatically.

```
1 connections:
2  - type: bridge
3    peers:
4     - name: localhost
5     - name: anarouter
6       b_ip_address:
7        - "inet6 2001:a98:13:4000::1 prefixlen 126"
8        - "inet 10.10.10.1/24"
9       n_ip_address:
```

```
10          - "inet6 2001:a98:13:4000::2 prefixlen 126"
11          - "inet 10.10.10.2/24"
12
13    - type: direct
14      bandwidth: 10mbps
15      delay: 1ms
16      downstream_ber: 2
17      peers:
18       - name: anarouter
19         ip_addresses:
20          - "inet6 2001:a98:13:4000::5 prefixlen 126"
21          - "inet 10.10.20.1/24"
22       - name: ituh
23         ip_addresses:
24          - "inet6 2001:a98:13:4000::6 prefixlen 126"
25          - "inet 10.10.20.2/24"
26
27    - type: bridge
28      peers:
29       - name: comur1
30         ip_addresses:
31          - "inet6 2001:a98:13:4004::1 prefixlen 64"
32          - "inet 10.10.70.1/24"
33         prefix: "2001:a98:13:4004::/64"
34       - name: yavuz
35         ip_addresses:
36          - "inet6 2001:a98:13:4004::200 prefixlen 64"
37          - "inet 10.10.70.2/24"
38       - name: emre
39         ip_addresses:
40          - auto6
41          - "inet 10.10.70.3/24"
```

Listing 6.3: Kovan Configuration Connections section

6.4 Services

Services field defines the services that will be run on virtual devices after network is established. Service definition has a simple template:

- **name**: The name of the service. This field is very important for softflowd services. If you want to use softflowd on all routers you must use exactly "softflowd" name.
- **node**: This parameter defines the node class that service will run on. Node parameter may be a device name such as monitor_host or special keyword defining node groups which are all-nodes and all-routers.
- **command**: This is the command required to run the service. Special values such as IP6 or IP4 can be used in commands parameter. If IP6 is used, IPv6 address of the corresponding device is replaced with this value in execution mode.

Listing 6.4 is taken from example configuration file. In this example syslogd service is configured to run on monitor_host and yavuz devices with given pid files. softflowd service will run on all routers. kovan_dns service is configured to run on yavuz device with the IP6 address of the yavuz node.

```
services:
 - name: syslogd
   node: monitor_host
   command: syslogd -P monitor.pid
 - name: syslogd
   node: yavuz
   command: syslogd -P yavuz.pid
 - name: softflowd
   node: all-routers
   command: /usr/local/sbin/softflowd
 - name: kovan_dns
   node: yavuz
   command: /usr/local/kovan/bin/kovan\_dns -h {IP6} -p 53 -c /usr/local/kovan
       /etc/dns/named.conf  -d /usr/local/kovan/etc/dns.pid
```

Listing 6.4: Kovan Configuration Services section

6.5 Example Topologies

In this section three example topologies are given to clarify configuration mechanism. Three topologies with different sizes will be represented.

Figure 6.1 show the graph of the small example topology. In this topology only one virtual device is created and connected to Base System.

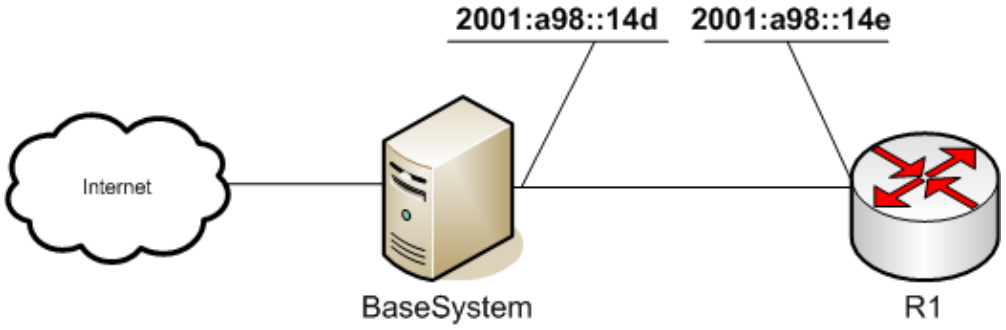

Figure 6.1: Small example Topology

Listing 6.5 shows the configuration file of the topology given in figure 6.1. In jail_roots section, only the definition of “router” is compulsory because a router device is created. Defining other types node, monitor and blackhole is not necessary. In devices section two devices are defined. First device has a special name “localhost” which means that this device is BaseSystem. If the device name is “localhost” Kovan does not create a virtual device but uses BaseSystem instead. In the definition of localhost, static route for virtual network is added under default_route topic. Connections section has only one definition which connects virtual router to BaseSytem, bridge connection is used for this connection.

```
---
physical\_ether: re0
kovan\_dir: /usr/local/kovan
```

```
jail\_roots:
 router: /usr/local/kovan/router
 node: /usr/local/kovan/node
 monitor: /usr/local/kovan/monitor
 blackhole: /usr/local/kovan/blackhole

devices:
 - name: localhost
   type: router
   default_route:
    - "-inet6 -net 2001:a98:14::/48 2001:a98:14::E"

 - name: R1
   type: router
   distribute: 1
   default_route:
    - "-inet6 default 2001:a98:14::D"

connections:

 - type: bridge
   peers:
    - name: localhost
    - name: R1
      b_ip_address:
       - "inet6 2001:a98:14::D prefixlen 126"
      n_ip_address:
       - "inet6 2001:a98:14::E prefixlen 126"
```

Listing 6.5: Small Example Configuration

Figure 6.2 show the graph of the medium example topology. In medium topology there are three more virtual devices are created and connected in addition to small topology.

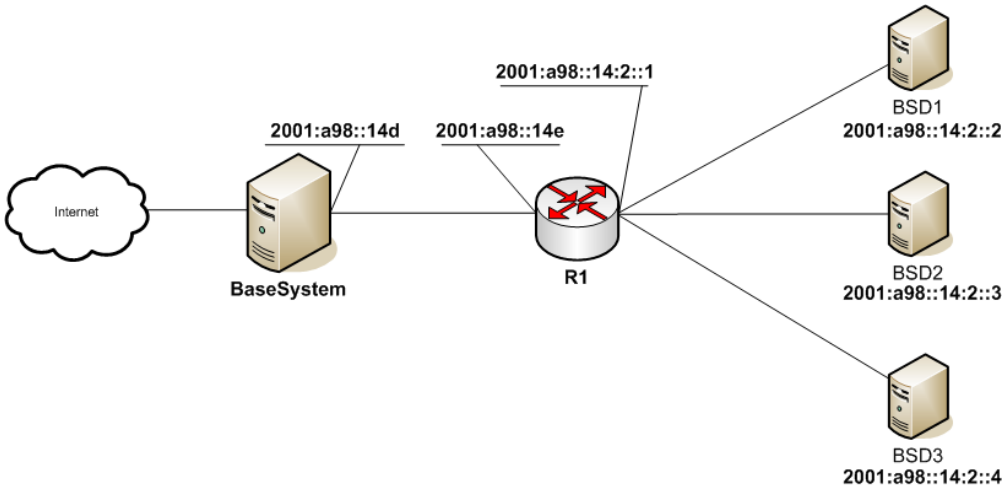

Figure 6.2: Medium example Topology

Listing 6.6 shows the configuration file of the topology given in figure 6.2. In addition to small topology three virtual hosts(BSD1, BSD2, BSD3) are defined and connected. Three devices are connected to R1 router using a bridge connection.

```
---
physical_ether: re0
kovan_dir: /usr/local/kovan

jail_roots:
 router: /usr/local/kovan/router
 node: /usr/local/kovan/node
 monitor: /usr/local/kovan/monitor
 blackhole: /usr/local/kovan/blackhole

devices:
 - name: localhost
   type: router
   default_route:
    - "-inet6 -net 2001:a98:14::/48 2001:a98:14::E"

 - name: R1
   type: router
   distribute: 1
   default_route:
    - "-inet6 default 2001:a98:14::D"

 - name: BSD1
   type: node

 - name: BSD2
   type: node

 - name: BSD3
   type: node

connections:

 - type: bridge
   peers:
    - name: localhost
    - name: R1
      b_ip_address:
       - "inet6 2001:a98:14::D prefixlen 126"
      n_ip_address:
       - "inet6 2001:a98:14::E prefixlen 126"

 - type: bridge
   peers:
    - name: R1
      ip_addresses:
       - "inet6 2001:a98:14:2::1 prefixlen 64"
      prefix: "2001:a98:14:2::/64"
```

```
  - name: BSD1
    ip_addresses:
     - "inet6 2001:a98:14:2::3 prefixlen 64"
  - name: BSD2
    ip_addresses:
     - "inet6 2001:a98:14:2::4 prefixlen 64"
  - name: BSD3
    ip_addresses:
     - "inet6 2001:a98:14:2::5 prefixlen 64"
```

Listing 6.6: Medium Example Configuration

There isn't any service defined on small and medium topologies only network is defined and you can test configuration via ping/traceroute commands. On the other hand in the last example, large topology (figure 6.3), has several services defined.

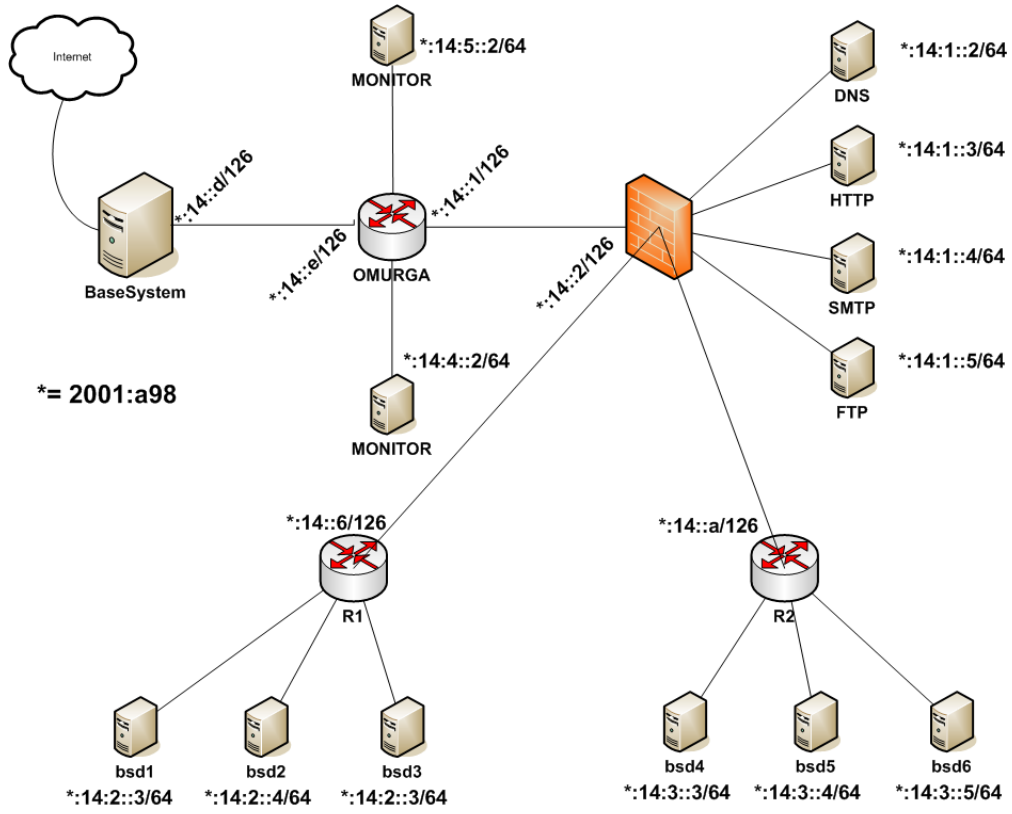

Figure 6.3: Large example Topology

Large topology defines three routers, one firewall, four servers and six clients. Different from small and medium examples, in large topology direct connections are used between routers and firewalls. In the last section of configuration file services are defined on virtual devices.

```
---
physical_ether: re0
kovan_dir: /usr/local/kovan
```

```
jail_roots:
 router: /usr/local/kovan/router
 node: /usr/local/kovan/node
 monitor: /usr/local/kovan/monitor
 blackhole: /usr/local/kovan/blackhole

devices:
 - name: localhost
   type: router
   default_route:
    - "-inet6 -net 2001:a98:14::/48 2001:a98:14::E"

 - name: OMURGA
   type: router
   distribute: 1
   default_route:
    - "-inet6 default 2001:a98:14::D"

 - name: FW
   type: router
   distribute: 0

 - name: R1
   type: router
   distribute: 0

 - name: R2
   type: router
   distribute: 0

 - name: MONITOR
   type: monitor
   distribute: 0
   default_route:
    - "-inet6 default 2001:a98:14:5::1"
   nfsen_start_port: 9000

 - name: BLACKHOLE
   type: blackhole
   default_route:
    - "-inet6 -net 2001:a98:14::/48  2001:a98:14:4::1"

 - name: BSD1
   type: node
 - name: BSD2
   type: node
 - name: BSD3
   type: node
 - name: BSD4
   type: node
```

```
  - name: BSD5
    type: node
  - name: BSD6
    type: node
  - name: DNS
    type: node
  - name: HTTP
    type: node
  - name: FTP
    type: node
  - name: SMTP
    type: node

connections:

  - type: bridge
    peers:
     - name: localhost
     - name: OMURGA
       b_ip_address:
        - "inet6 2001:a98:14::D prefixlen 126"
       n_ip_address:
        - "inet6 2001:a98:14::E prefixlen 126"

  - type: bridge
    peers:
     - name: localhost
     - name: BLACKHOLE
       n_ip_address:
        - "inet6 2001:a98::173 prefixlen 125"

  - type: direct
    peers:
     - name: OMURGA
       ip_addresses:
        - "inet6 2001:a98:14:5::1 prefixlen 64"
     - name: MONITOR
       ip_addresses:
        - "inet6 2001:a98:14:5::2 prefixlen 64"

  - type: direct
    peers:
     - name: OMURGA
       ip_addresses:
        - "inet6 2001:a98:14:4::1 prefixlen 64"
     - name: BLACKHOLE
       ip_addresses:
        - "inet6 2001:a98:14:4::2 prefixlen 64"

  - type: direct
    peers:
```

```
    - name: OMURGA
      ip_addresses:
       - "inet6 2001:a98:14::1 prefixlen 126"
    - name: FW
      ip_addresses:
       - "inet6 2001:a98:14::2 prefixlen 126"

- type: direct
  peers:
   - name: FW
     ip_addresses:
      - "inet6 2001:a98:14::5 prefixlen 126"
   - name: R1
     ip_addresses:
      - "inet6 2001:a98:14::6 prefixlen 126"

- type: direct
  peers:
   - name: FW
     ip_addresses:
      - "inet6 2001:a98:14::9 prefixlen 126"
   - name: R2
     ip_addresses:
      - "inet6 2001:a98:14::a prefixlen 126"

- type: bridge
  peers:
   - name: R1
     ip_addresses:
      - "inet6 2001:a98:14:2::1 prefixlen 64"
     prefix: "2001:a98:14:2::/64"

   - name: BSD1
     ip_addresses:
      - "inet6 2001:a98:14:2::3 prefixlen 64"
   - name: BSD2
     ip_addresses:
      - "inet6 2001:a98:14:2::4 prefixlen 64"
   - name: BSD3
     ip_addresses:
      - "inet6 2001:a98:14:2::5 prefixlen 64"

- type: bridge
  peers:
   - name: R2
     ip_addresses:
      - "inet6 2001:a98:14:3::1 prefixlen 64"
     prefix: "2001:a98:14:3::/64"

   - name: BSD4
     ip_addresses:
```

```
        - "inet6 2001:a98:14:3::3 prefixlen 64"
      - name: BSD5
        ip_addresses:
        - "inet6 2001:a98:14:3::4 prefixlen 64"
      - name: BSD6
        ip_addresses:
        - "inet6 2001:a98:14:3::5 prefixlen 64"

  - type: bridge
    peers:
      - name: FW
        ip_addresses:
        - "inet6 2001:a98:14:1::1 prefixlen 64"
        prefix: "2001:a98:14:1::/64"

      - name: DNS
        ip_addresses:
        - "inet6 2001:a98:14:1::2 prefixlen 64"
      - name: HTTP
        ip_addresses:
        - "inet6 2001:a98:14:1::3 prefixlen 64"
      - name: SMTP
        ip_addresses:
        - "inet6 2001:a98:14:1::4 prefixlen 64"
      - name: FTP
        ip_addresses:
        - "inet6 2001:a98:14:1::5 prefixlen 64"

services:
  - name: softflowd
    node: all-routers
    command: /usr/local/sbin/softflowd
  - name: cron
    node: MONITOR
    command: /etc/rc.d/cron onestart
  - name: snmpd
    node: all-routers
    command: /usr/local/sbin/snmpd -c /etc/snmpd.conf udp6:161
  - name: snmpd
    node: BLACKHOLE
    command: /usr/local/sbin/snmpd -c /etc/snmpd.conf udp6:161
  - name: syslogd
    node: HTTP
    command: syslogd -P syslog.pid
  - name: logger
    node: HTTP
    command: tail -f /var/log/messages | grep kovan | logger -h 2001:a98
        :14:5::2 -P 510 &
  - name: kovan_dn\_6
    node: DNS
    command: /usr/local/kovan/bin/kovan\_dns -h {IP6} -p 53 -c /usr/local/kovan
```

```
    /etc/dns/named.conf  -d /usr/local/kovan/etc/dns6.pid
- name: kovan_mail_6
  node: SMTP
  command: /usr/local/kovan/bin/kovan\_smtp -h {IP6} -p 25 -c /usr/local/
    kovan/etc/smtp/ipv6go\_mail.conf  -d /usr/local/kovan/etc/smtp6.pid
- name: kovan_http_6
  node: HTTP
  command: /usr/local/kovan/bin/kovan\_http -h {IP6} -p 80 -w /usr/local/
    kovan/etc/www/  -d /usr/local/kovan/etc/http6.pid
- name: log_sql
  node: MONITOR
  command: /usr/local/etc/rc.d/mysql-server onestart
- name: syslog
  node: MONITOR
  command: /usr/local/sbin/syslog-ng
- name: flow_logger
  node: BLACKHOLE
  command: /usr/local/kovan/bin/kovan\_dump -i eth0 -f 'ip6 and host not
    2001:a98::173' |  logger -h 2001:a98:14:5::2 -P 510 &
```

Listing 6.7: Large Example Configuration

6.6 Hardcoded Passwords

src/kovan$_{h}ost/etc/snmpd.conf syslocation"KovanNetwork"syscontact"Onur"sysservices0rocommunity6public$
src/install.sh jexec $JAILIDcsh-c"echoC\"REATEUSER'logger'@'localhost'IDENTIFIEDBY'231905'|mysql-uroot"jexec$JAILID csh -c "echo G\"RANT ALL ON kovan.* TO 'logger'@'localhost'—¨ mysql -uroot" jexec $JAILIDcsh - c"mysqladmin - urootpassword231905"

Bibliography

[1]

[2]

[3]

Chapter 7

Running Kovan

Once the Kovan installed according to steps explained at section 5, it can be run to create virtual honeypots. This section covers the possible user scenarios of Kovan. Kovan is installed under a prefix given installation process. User scenarios assumes Kovan is installed under default installation path, "/usr/local/kovan".

Below shows the "kovan" command's basic help. "kovan" is the main tool to generate/destroy/save/load virtual honeypot topologies.

```
root# ./kovan
Configuration file is not given
usage: kovan --conf conf_file --list
       kovan --conf conf_file --execute
       kovan --conf conf_file --remove --list {destroys system}
       kovan --conf conf_file --remove --execute {destroys system}
```

7.1 Verifying Kovan configuration

Section 6 explains Kovan configuration in detail. An example kovan configuration file is resides in /usr/local/kovan/etc/ directory. New configuration files can be put any directory with read access.

Kovan makes configuration checks whenever possible automatically. But because it generates complex network topologies all checks can not be done automatically, in some cases a human eye is needed. Therefore, Kovan provides a "list" mode where all generated commands are printed to screen for manual check.

```
1  root# ./kovan -c ../etc/kovanTopology.conf -l
2  kldload ng_ether
3  kldload ng_bridge
4  kldload ng_eiface
5  kldload ng_iface
6  mount_nullfs -o ro /usr/jails/basejail  /usr/local/kovan/router/basejail/
7  ...
8  jail -c vnet name=onur host.hostname=onur path=/usr/local/kovan/node persist
9  ...
10 # It is normal to see "jexec: jail 'xxx' not found" errors in this execution
       mode!
11 jexec: jail "yavuz" not found
12 ...
13 jexec onur /usr/local/kovan/bin/kovan_http -h  -p 80 -w /usr/local/kovan/www/
       -d /usr/local/kovan/etc/http.pid
```

Listing 7.1: Listing Kovan configuration

It is important to note that Kovan list mode is not able to list information based on previous commands. For example (figure 7.1, in services part a http service is defined on jail onur. But node "onur" is not created during list mode, just the command required to create node "onur" is created (line 8). Therefore list mode can not determined the IP address of the HTTP service that will be bound to (line 13). The warning message on line 9 is generated for this case.

7.2 Starting and stopping Kovan

Unlike common applications, Kovan is not a single process that can be executed/killed using a pid value. Kovan is a system consists of several processes and system objects. Kovan start routines creates necessary objects(nodes,links ..etc), connects them and executes services over virtual topology. Listing 7.2 is an example start process of Kovan. In this example, kovanTopology.conf file is used as configuration file.

```
1  root# ./kovan -c ../etc/kovanTopology.conf -e
2  command: kldload ng_ether
3  command finished
4  command: kldload ng_bridge
5  command finished
6  command: kldload ng_eiface
7  command finished
8  command: kldload ng_iface
9  command finished
10 command: mount_nullfs -o ro /usr/jails/basejail  /usr/local/kovan/router/
       basejail/
11 command finished
12 ...
13 command: jail -c vnet name=anarouter host.hostname=anarouter path=/usr/local/
       kovan/router allow.raw_sockets  persist
14 ...
15 command: jexec comuh /usr/local/sbin/snmpd -c /etc/snmpd.conf udp6:161
16 command finished
17 ...
18 command: jexec onur /usr/local/kovan/bin/kovan_http -h 2001:a98:13:4004::201 -
       p 80 -w /usr/local/kovan/www/ -d /usr/local/kovan/etc/http.pid
19 command finished
```

Listing 7.2: Starting Kovan with an example configuration

Kovan stop routine deletes all resources generated at start process. Start process tracks all created objects, nodes, links, static routes ..etc and writes them to a temporary file named /usr/local/kovan/topology.tmp. When Kovan stop command is called the temporary file is read and resources are freed in a suitable manner (listing 7.3).

```
1  root# ./kovan -c ../etc/kovanTopology.conf -e -r
2  command: route delete -inet6 -net 2001:a98:13:4000::/60 2001:a98:13:4000::2
3  delete net 2001:a98:13:4000::/60: gateway 2001:a98:13:4000::2
4  command finished
5  command: route delete -inet -net 10.10.10.0/24 10.10.10.2
6  delete net 10.10.10.0: gateway 10.10.10.2
7  command finished
8  command: jail -r anarouter
9  command finished
10 ...
```

```
11 ...
```

Listing 7.3: Stopping currently running Kovan

Two Kovan instances can not be run at the same time o the same system. Kovan reveals the currently running system by checking the existence of /usr/local/kovan/topology.tmp file. Trying to start a Kovan instance when another one running results in error as shown at listing 7.4.

```
1 root# ./kovan -c ../etc/kovanTopology.conf -e
2 ! /usr/local/kovan/topology.tmp exists
3 Destroy previous configuration before establishing new one
4 usage:  kovan --conf conf_file --list
5         kovan --conf conf_file --execute
6         kovan --conf conf_file --remove --list {destroys system}
7         kovan --conf conf_file --remove --execute {destroys system}
```

Listing 7.4: Starting Kovan error

7.3 Save/load Kovan state

Chapter 8

Advanced Topics

8.1 Connecting KVM images (Windows, Solaris, Centos...)

You can attach QEMU/KVM virtual machine images to Kovan framework to improve your honeynet interaction level. Before attacking a host, intruders use active/passive fingerprinting tools to detect the remote OS and services. This phase of an attack is called reconnaissance. By using this technique an experienced intruder may distinguish the real OS and services to emulated ones. To make Kovan honeynet look more realistic, you can attach QEMU images of different operating systems to Kovan framework. For example, suppose you want to simulate the Windows running student PC lab in kovan environment. You can do this either by:

1. Creating Kovan nodes with simulated windows services
2. Using real windows QEMU images under Kovan framework.

If you use the latter one, it is more difficult to recon weather the remote network /OS /serivices are emulated or real. In order to attach QEMU images to Kovan you need to create an epair, a tap and a bridge interface. For connecting QEMU to Kovan framework tap and epair interfaces are attached to bridge as described in figure 8.1. Tap interface is used for QEMU to connect the Kovan framework and epair interface is used for Kovan to connect the QEMU image. Bridge is used for connecting two interfaces.

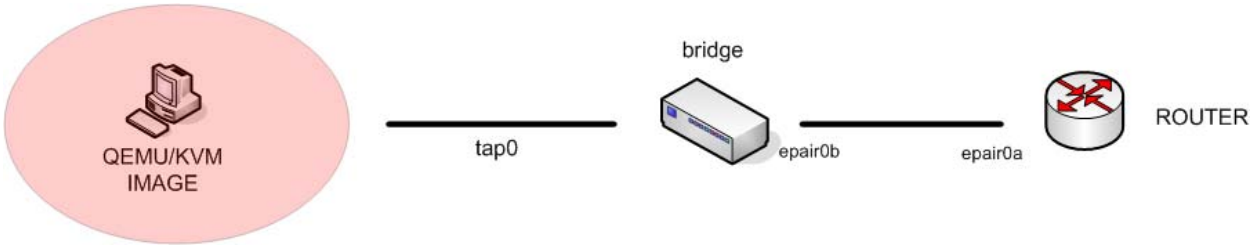

Figure 8.1: QEMU VM attach to Kovan Router

Attaching an QEMU image to Kovan is not difficult, for single installation you need to follow the 10 steps.

1. Setup your FreeBSD base system to run QEMU images as written in section 5.5.1 FreeBSD QEMU/KVM Setup.
2. Install your QEMU host

3. Desing a network topolgy
4. Prepare the IPv4/IPv6 addresses according to your topology
5. Configure Router that QEMU attached
6. Prepare a kovan.conf
7. Decide where you want to plug your QEMU image to Kovan inftrastructure
8. Run Kovan
9. Attach QEMU image to Kovan
10. Run QEMU image

Let us give an example for better understanding of the topic, we use the example1.conf as a Kovan configuration file which stands the topolgy in Figure 8.2

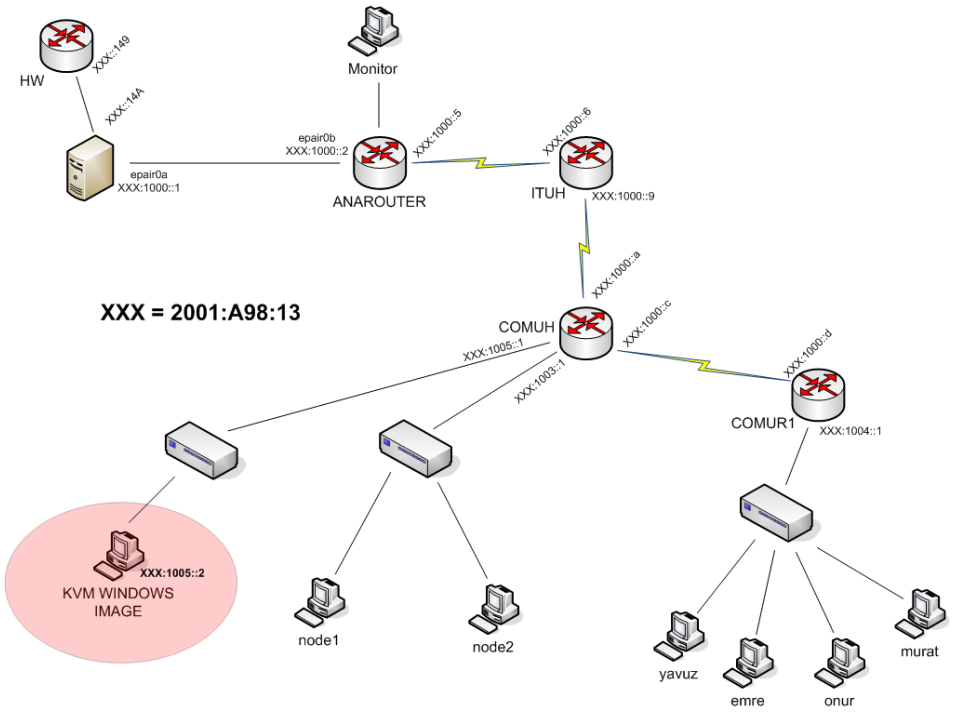

Figure 8.2: QEMU VM Example Topology (example1.conf)

1. Follow section 5.5.1.
2. Follow QEMU documentation. For the rest of the example we assume QEMU image file is win7.img running Microsoft Windows 7
3. For step 3-7 use example1.conf in the examples directory.
4. Run kovan
5. Attach QEMU images to COMUH router as described in Figure 8.3

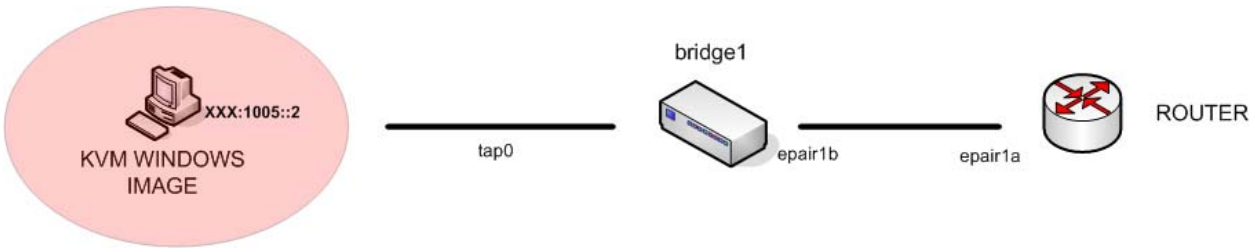

Figure 8.3: QEMU Windows 7 attached to COMUH Router

(a) Create bridge, tap, epair interfaces

```
root# ifconfig bridge create
brigde1
root# ifconfig tap create
tap0
root# ifconfig epair create
epair1a
```

(b) Attach epair1b to bride

```
root# ifconfig bridge1 addm epair1b
```

(c) Attach epair1a to Kovan COMUH Router

```
root# ifconfig epair1a vnet comuh
```

(d) Attach tap interface to bridge

```
root# ifconfig bridge1 addm tap0
```

(e) Give epair1a IPv4&IPv6 addreses

```
root# ifconfig bridge1 up
root# ifconfig epair1b up
root# jexec comuh ifconfig epair1a up
root# jexec comuh ifconfig epair1a 10.10.80.1 netmask 255.255.255.0 up
root# jexec comuh ifconfig epair1a inet6 2001:a98:13:1005::1 prefixlen 64 up
root# ifconfig tap0 up
```

The routing should congfigured automatically

6. Start Kovan FreeBSD image with 1 Gbyte of memory 1 core on vnc 1

```
qemu-system-x86_64 -m 1024 -smp 1  -net nic -net tap,name=tap0, -hda win7.img -boot c -vnc :1
```

7. Connect you windows image using your favorite VNC viewer and configure your Windows IPv6/IPv4 address

8.2 Changing Network Link Parameters (BER, Bandwidth, Delay)

Listing 8.1: Connecting two ng_eiface node with pipe

Chapter 9

Security

9.1 Security considerations

```
#!/bin/bash

IPTABLES="/sbin/iptables"
IP6TABLES="/sbin/ip6tables"

#### Bridge Interface
BRIDGE=br0
### KVM Linux Server Interfaces
MNG_IP="193.140.30.250"
MNG_IP6="2001:a98::16A/125"

### System administrators net
MNG_NET="193.140.94.0/24"
MNG_NET6="2001:a98:12::/64"

### VNC Port Range
VNC_RANGE="5900:5920"

#
# Flush (-F) all specific rules
#
$IPTABLES -F INPUT
$IPTABLES -F FORWARD
$IPTABLES -F OUTPUT
$IPTABLES -F -t nat

#### Flush Rules For IPv6
$IP6TABLES -F INPUT
$IP6TABLES -F FORWARD
$IP6TABLES -F OUTPUT

### Accept Everyting
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT
```

```
$IPTABLES -P OUTPUT ACCEPT

### Accept Everyting For IPv6

$IP6TABLES -P INPUT ACCEPT
$IP6TABLES -P FORWARD ACCEPT
$IP6TABLES -P OUTPUT ACCEPT


### Allow management IP address SSH and VNC connection
$IPTABLES -A INPUT -p tcp -i $BRIDGE -s $MNG_NET -d $MNG_IP -m multiport --
    dports 22,$VNC_RANGE -m state --state NEW -j ACCEPT
$IP6TABLES -A INPUT -p tcp -i $BRIDGE -s $MNG_NET6 -d $MNG_IP6 -m multiport
    --dport 22,$VNC_RANGE -m state --state NEW -j ACCEPT
$IPTABLES -A INPUT -p udp -i $BRIDGE -s $MNG_NET -d $MNG_IP -m multiport --
    dports 22,$VNC_RANGE -m state --state NEW -j ACCEPT
$IP6TABLES -A INPUT -p udp -i $BRIDGE -s $MNG_NET6 -d $MNG_IP6 -m multiport
    --dport 22,$VNC_RANGE -m state --state NEW -j ACCEPT

### Deny all other connection to SSH and VNC
$IPTABLES -A INPUT -p tcp -i $BRIDGE -d $MNG_IP -m multiport --dports 22,
    $VNC_RANGE -m state --state NEW -j DROP
$IP6TABLES -A INPUT -p tcp -i $BRIDGE -d $MNG_IP6 -m multiport --dports 22,
    $VNC_RANGE -m state --state NEW -j DROP
$IPTABLES -A INPUT -p udp -i $BRIDGE -d $MNG_IP -m multiport --dports 22,
    $VNC_RANGE -m state --state NEW -j DROP
$IP6TABLES -A INPUT -p udp -i $BRIDGE -d $MNG_IP6 -m multiport --dports 22,
    $VNC_RANGE -m state --state NEW -j DROP
```

Listing 9.1: Firewall Configuration

Chapter 10

Performance

10.1 Tuning Kovan for maximum performance

10.2 Large installation tweaks

Chapter 11

Integration With Other Software

11.1 Apache

11.2 Other Honeypots

11.3 Monitor

Chapter 12

Future Work

12.1 Defeating OS Fingerprinting

Chapter 13

Services

Kovan uses application layer services to attract attackers. Using a real service or fake service is completely depends on system administrator's needs. If real services like Apache, bind, postfix are used; a higher level of interaction occurs. The advantage of using real services and creating a higher level of interaction with an attacker is that attacker can not easily detect honeypot. On the other hand, providing a higher level of interaction increases the possibility of an attacker breaking your system. Contrary, using fake systems decreases interaction level and increases the honeypot detection possibility.

Kovan provides four frequently used services' lightweight implementation. HTTP, DNS, SMTP and FTP services are implemented using a service framework named Generically. All services and framework are implemented in C for efficiency and low resource consumption. These services are just examples, administrator can implement his/her own services or use a real service instead.

13.1 Generic Server

Network programmers deal with lots of common issues such as reading/writing packets, checking integrity, checking packet acknowledgment ... etc during network application development. Fortunately, operating systems have well defined layered network architectures and provides standard interfaces to developers for network operations. Developers can design applications on application layer using lower layer interfaces. Although common interfaces make the job simpler, there is need for a more skillful framework while developing complex network services such as HTTP, SMTP, DNS.. etc . In addition to basic socket/connection operations, developers have to think on os related issues such as multi-threaded architecture, shared memory management and resource consumption. Kovan Generic Server (Kovan_GS) is designed to be a base platform for network server development.

Kovan_GS conducts sockets/threads related issues and provides only necessary information to upper application. Application developer does not have to think about common operation such as open/close socket, read/write sockets, listen on sockets and create/destroy processes. Developer only gets source/destination addresses and client socket descriptors from Kovan_GS. As a result, implementing a web server with Kovan_GS becomes just processing incoming messages and writing back suitable answers.

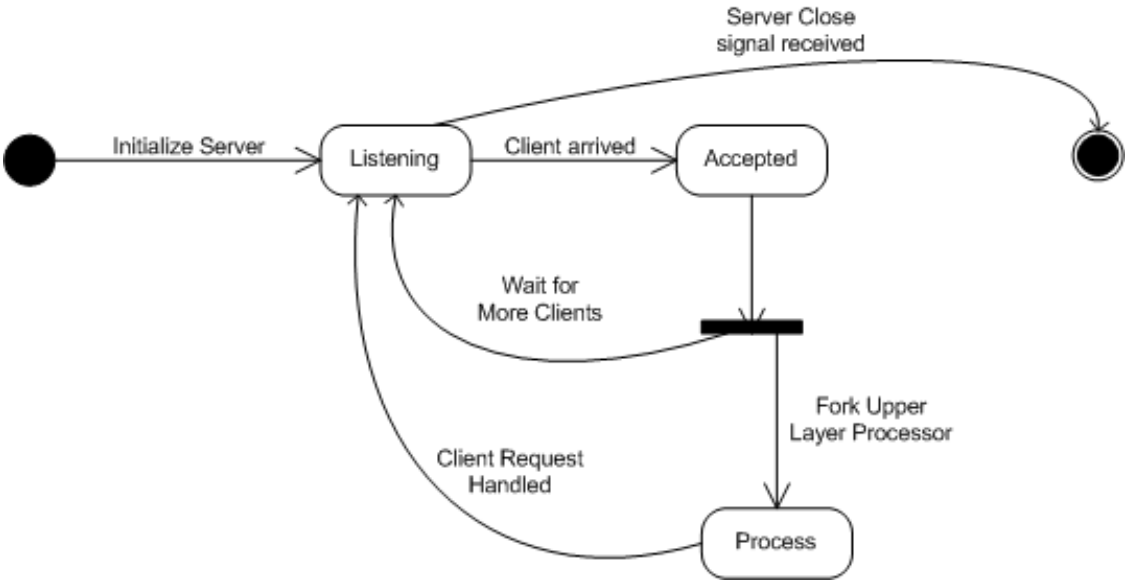

Figure 13.1: Generic Server's execution states

Figure 13.1 explains general working principle of the framework. Kovan_GS accepts connections, generates worker processes and calls upper layer methods inside processes.

13.2 HTTP

Kovan HTTP service is designed top of the Kovan_GS framework. Kovan_GS framework does preprocessing and the postprocessing of the connection as can be seen at figure 13.2. When all the common preprocessing is done, necessary data is passed to upper layer process, HTTP process in this case. HTTP service parses incoming TCP packet and acts according to request.

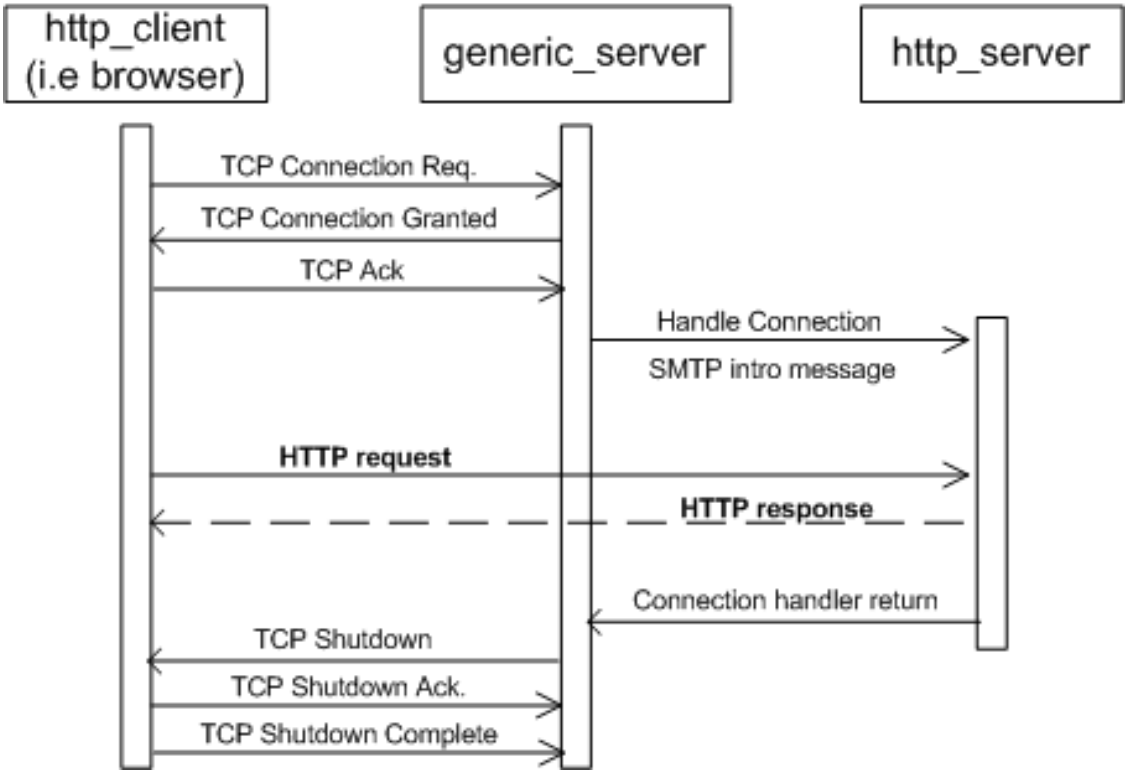

Figure 13.2: HTTP service sequence diagram

Figure 13.3 show HTTP service program flow. Kovan HTTP reads TCP packet line-by-line and extracts request type, URL and http version on the first step. Later steps gets keyword:value pairs from message. All extracted information is stored in request specific data structure. When TCP header is finished, HTTP process checks header validity. If request is a valid request, server tries to handle request (i.e send requested file). If requested file is exist it is send to client, else appropriate response message is send.

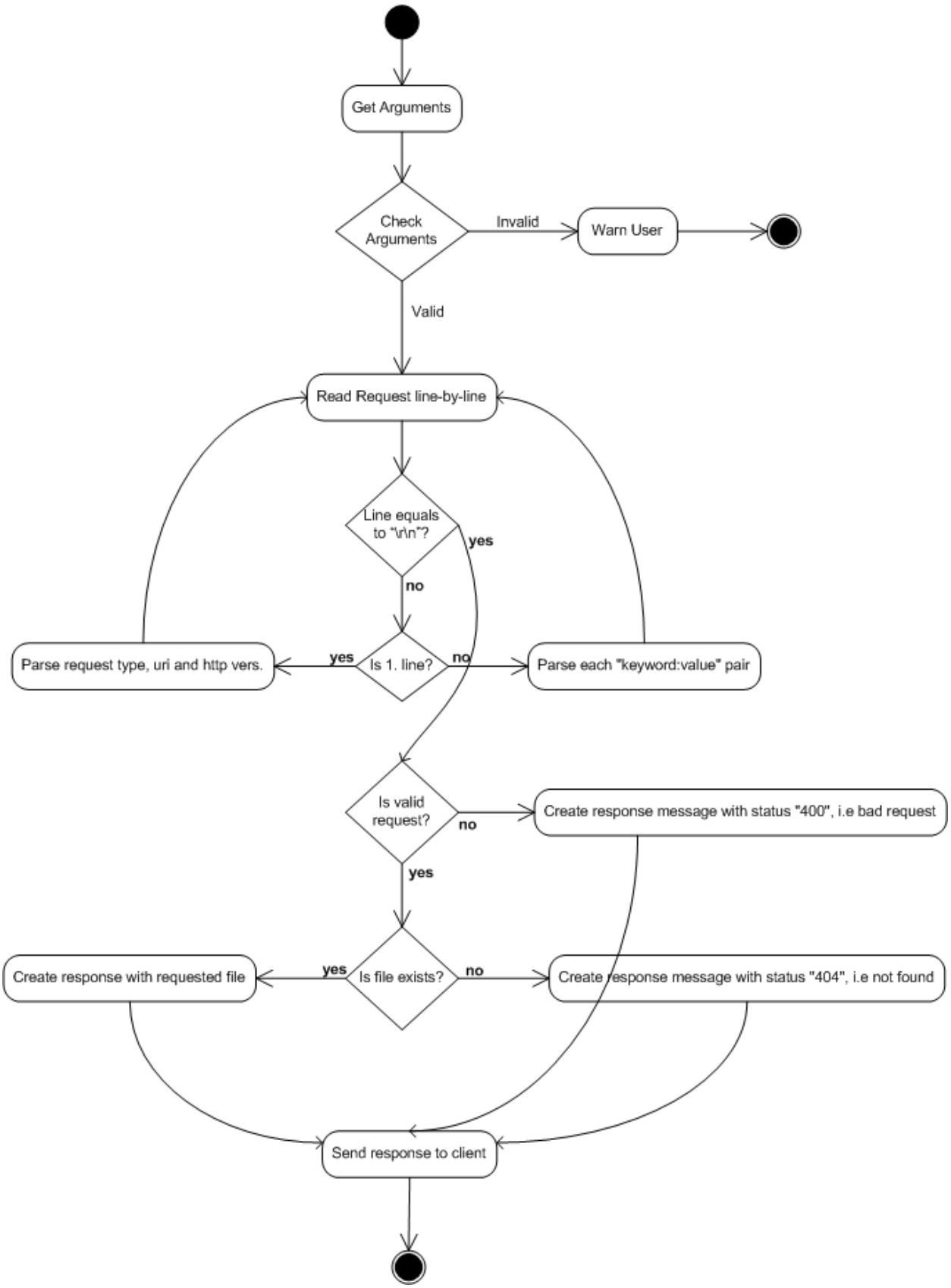

Figure 13.3: HTTP service flow diagram

kovan_http takes three must one optional arguments as shown below. If it is wanted to execute application in the background "-d pidfile" argument should be used. host and port arguments are the IP address and port number that the http service binds. webroot holds is the path for the web files.

```
root# ./kovan_http
usage: kovan_http [-d pidfile] -h host -p port -w webroot
```

13.3 DNS

Kovan DNS service is designed top of the Kovan_GS framework. DNS service works only on UDP and Kovan_GS framework does preprocessing and the postprocessing of UDP as can be seen at figure 13.4. UDP messages are send to upper layer processor and reply messages are send to client.

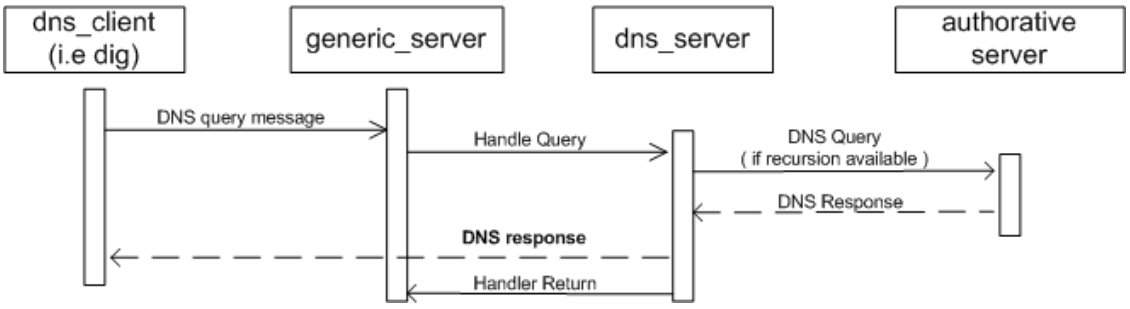

Figure 13.4: DNS service sequence diagram

Figure 13.5 show DNS program flow. On the first step, Kovan DNS reads configuration file and creates lookup tables of given domain names. If an error occurs during lookup table creation, program terminates with an error message send to syslog. On each DNS query, a worker process is generated to handle query request. All worker processes read the same lookup table but does not modify. If query request is not a supported type, an error reply is returned to client. If request type is supported, Kovan DNS starts lookup process.

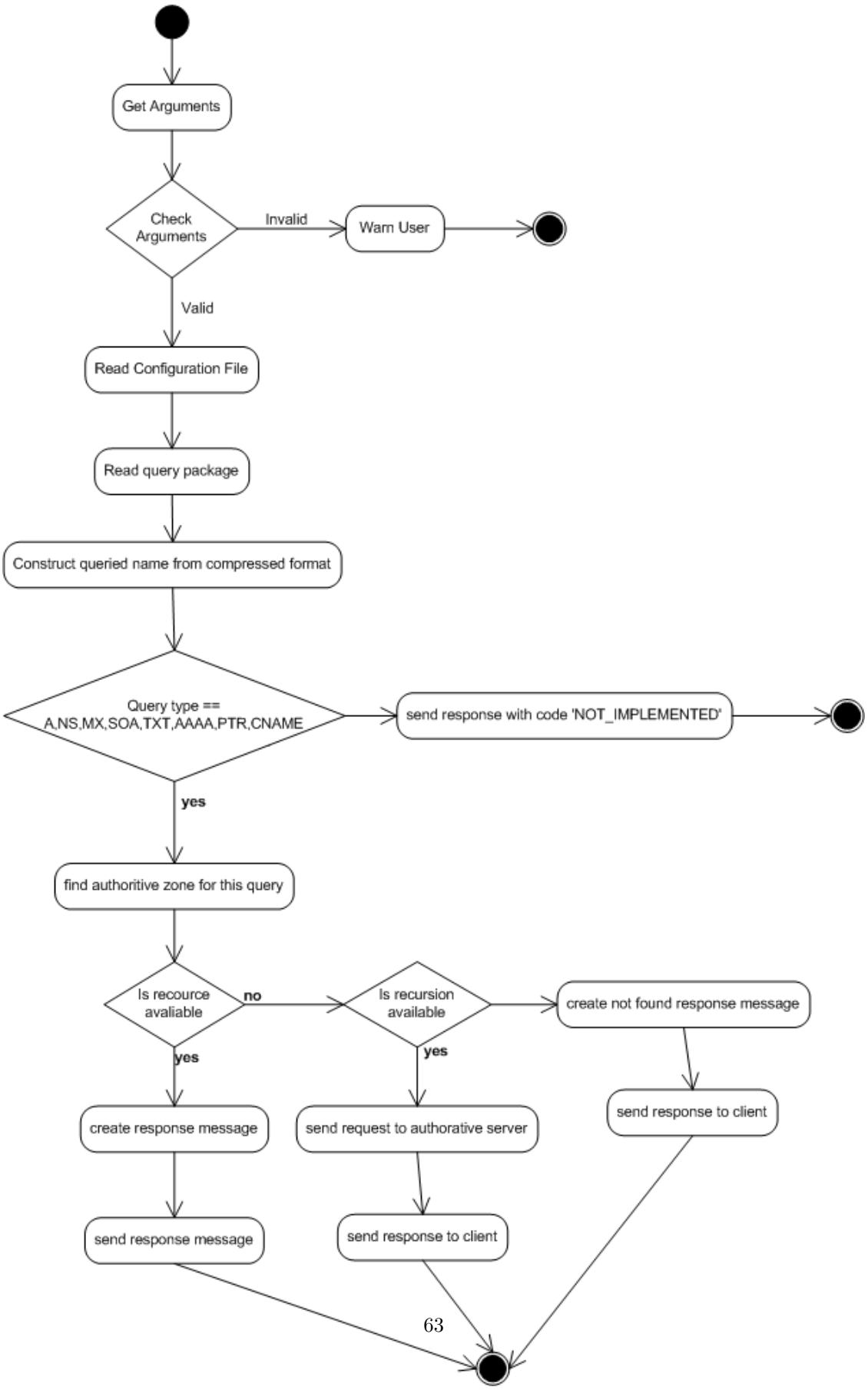

Figure 13.5: DNS service flow diagram

kovan_dns takes three must and two optional arguments as shown below. If it is wanted to execute application in the background "-d pidfile" argument should be used. In addition, -r flag should be given if a recursive dns is nedeed. host and port arguments are the IP address and port number that the doman name service binds.

```
root# ./kovan_dns
usage: kovan_dns [-d pidfile] -h host -p port -c config [-r recurse]
```

The name of the DNS configuration file can be given via "-c config" argument (This path should be a fullpath). Configuration file contains one line for each authorative domain. Each line has the form "domain - zonefile":

```
root# cat named.conf
"ipv6go.ipv6.net.tr." "ipv6go.ipv6.net.tr.conf"
```

Zonefiles have a similar format with real DNS zonefiles (wildcards are not allowed). A simple zone file can be seen in listing 13.1:

```
root# cat ipv6go.ipv6.net.tr.conf
$ORIGIN ipv6go.ipv6.net.tr.
$TTL 1h                 ; comments...
ipv6go.ipv6.net.tr. IN  SOA ns1.ipv6go.ipv6.net.tr. hostmaster.ipv6go.ipv6.net
    .tr. 2009082702 28800 7200 604800 600
ipv6go.ipv6.net.tr. NS     ns1.ipv6go.ipv6.net.tr.
ipv6go.ipv6.net.tr. MX     10 mail.ipv6go.ipv6.net.tr.
ipv6go.ipv6.net.tr. AAAA   2001:a98:14:1::3
www    AAAA   2001:a98:14:1::3
mail   AAAA   2001:a98:14:1::4
ns1    AAAA   2001:a98:14:1::2
ftp    AAAA   2001:a98:14:1::5

bsd1.computerscience     AAAA   2001:a98:14:2::3
bsd2.computerscience     AAAA   2001:a98:14:2::4
bsd3.computerscience     AAAA   2001:a98:14:2::5

bsd4.management          AAAA   2001:a98:14:3::3
bsd5.management          AAAA   2001:a98:14:3::4
bsd6.management          AAAA   2001:a98:14:3::5
```

Listing 13.1: Contents of a zone file

13.4 FTP

FTP service is implemented as a message server. It returns suitable answers to client but does not create/put/get any data from/to ftp server. Figure 13.6 shows simple sequence diagram of the FTP service.

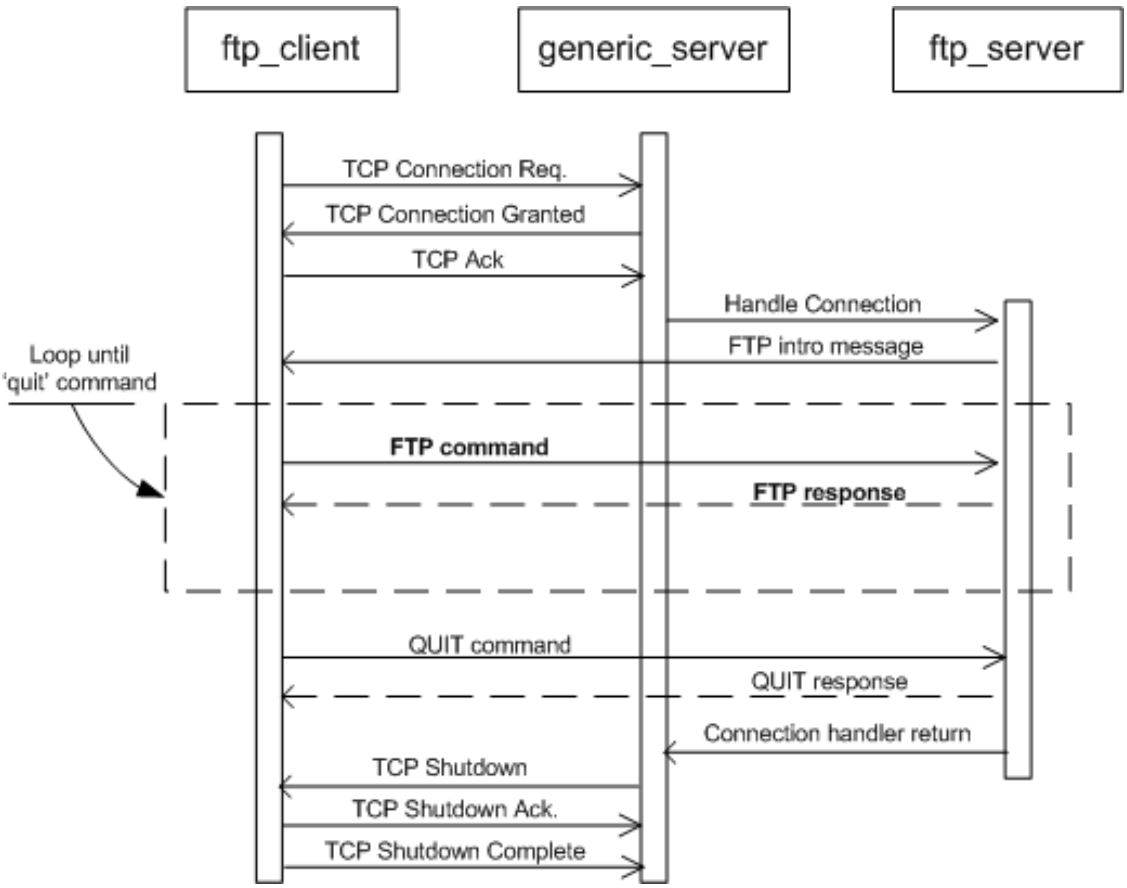

Figure 13.6: FTP service sequence diagram

kovan_ftp takes two must one optional arguments as shown below. If it is wanted to execute application in the background "-d pidfile" argument should be used. host and port arguments are the IP address and port number that the ftp service binds.

```
root# ./kovan_ftp
usage: kovan_ftp [-d pidfile] -h host -p port
```

13.5 SMTP

Kovan SMTP service is designed top of the Kovan_GS framework. Kovan_GS framework does preprocessing and the postprocessing of the connection as can be seen at figure 13.7. When all the common preprocessing is done, necessary data is passed to upper layer SMTP process in this case. SMTP service is an interactive service, it responds during tcp connection according to request messages.

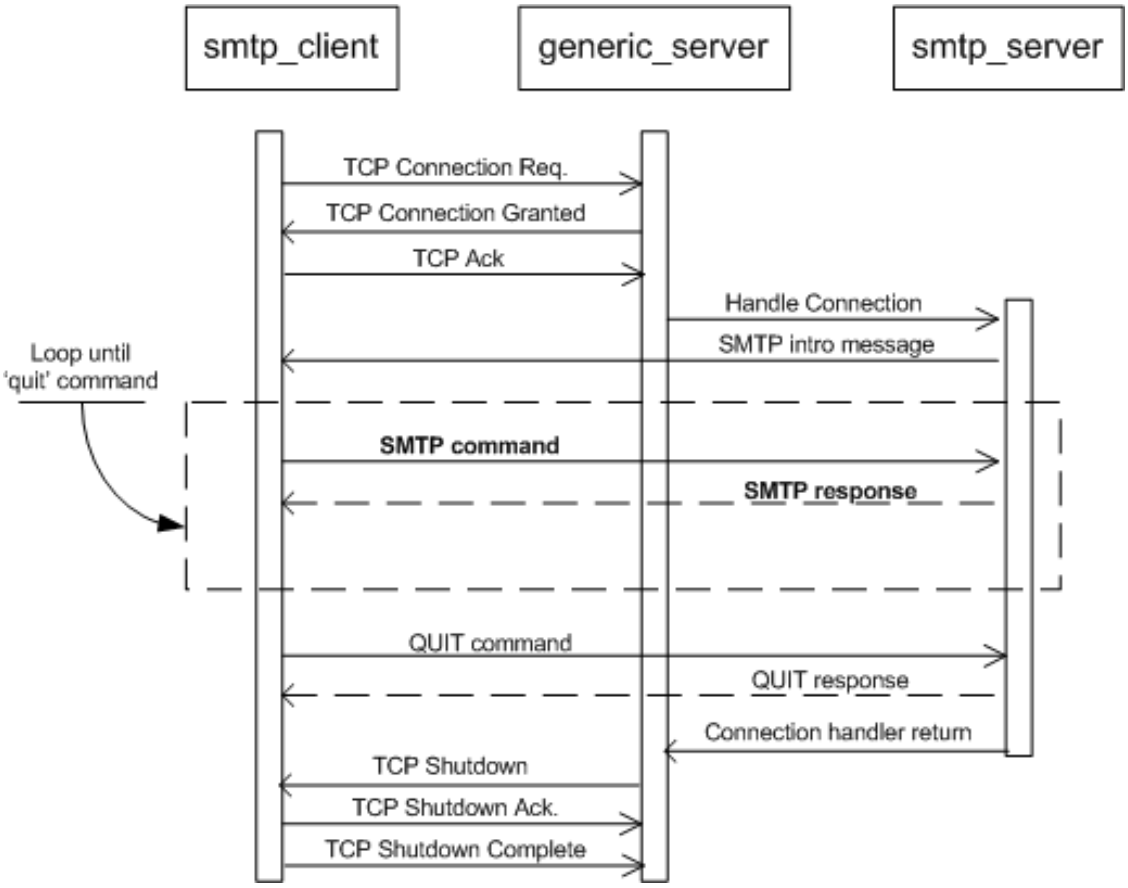

Figure 13.7: SMTP service sequence diagram

SMTP services has a complex program flow as shown on figure 13.8. Flow complexity is the result of interactive property and stateful architecture. Interactivity increases possible transmissions that can occur during program flow. In addition, stateful architecture adds state-to-state checks in program flow (i.e you should not jump to data state from initial state).

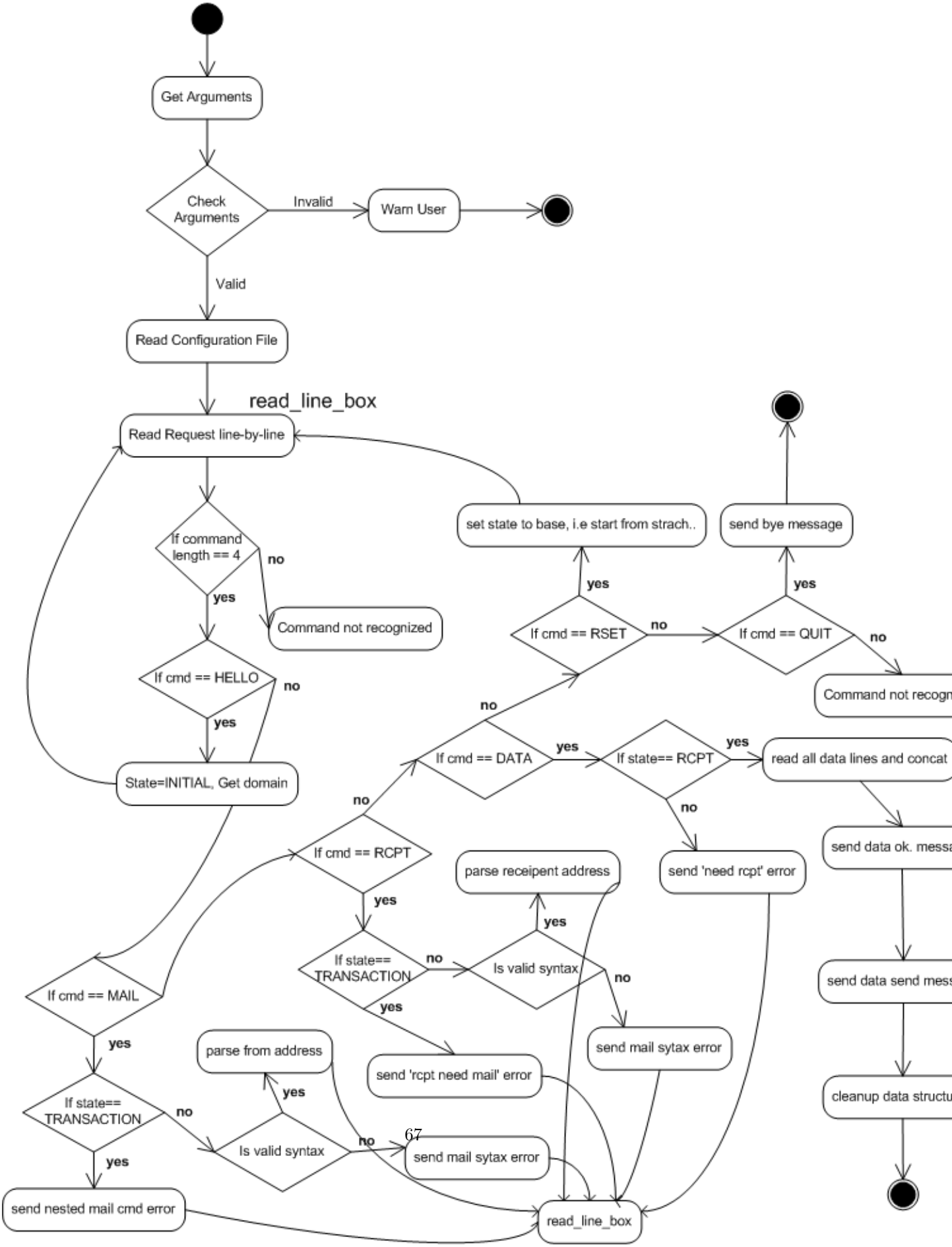

Figure 13.8: SMTP service flow diagram

kovan_smtp takes three must one optional arguments as shown below. If it is wanted to execute application in the background "-d pidfile" argument should be used. host and port arguments are the IP address and port number that the ftp service binds.

```
root# ./kovan_smtp
usage: kovan_smtp [-d pidfile] -h host -p port -c config
```

The name of the SMTP configuration file can be given via "-c config" argument. Configuration file contains one line for each response message definitions:

```
HELO_250 "250 mail.ulakbim.gov.tr"
MAIL_250 "250 ok"
HELO_FIRST_503 "503 5.5.1 Error: send HELO/EHLO first"
MAIL_SYNTAX_ERR "501 5.5.4 Syntax: MAIL FROM:<address>"
MAIL_NESTED_ERR "503 5.5.1 Error: nested MAIL command"
QUIT_BYE "221 2.0.0 Bye"
RCPT_NEED_MAIL "503 5.5.1 Error: need MAIL command"
RCPT_SYNTAX_ERR "501 5.5.4 Syntax: RCPT TO:<address>"
RCPT_OK "250 2.1.5 Ok"
DATA_NEED_RCPT "503 5.5.1 Error: need RCPT command"
DATA_START "354 End data with <CR><LF>.<CR><LF>"
DATA_OK "250 2.0.0 Ok: queued as 296F511E823"
CMD_NOT_RECOG "502 5.5.2 Error: command not recognized"
```

Listing 13.2: SMTP configuration file

Chapter 14

Logging

Kovan collects logs from services and black hole. Kovan services are designed to write error logs and access logs to /var/messages. A forwarder script is written to send all logs to monitor host continuously. Kovan services use a common log format shown below:

```
Nov 22 15:17:30 HTTP kovan_http: 2001:a98:14:5::2 1755
    2001:a98:14:1::3 80 tcp "GET / HTTP/1.0 404 0" 1290439050
```

Common log format has 10 fields which are: log date, source node, source service, source IP, source PORT, destination IP, destination PORT, used protocol, custom message, utc time. In Kovan, all service nodes use the same jail root thus they are using the same log device and writing to the same /var/log/messages file. Therefore, only one syslog instance can be executed for all services. As a result, second field of the common log format is same for all services logs, it is the name of the virtual node which executes syslog application.

In addition to Kovan service activities, blackhole flows are logged to monitor host. Blackhole logs has the form show below:

```
Nov 22 15:17:30 BLACKHOLE blackhole: 2001:a98:12::151 -1
    2001:a98::abab -1 icmp6 blackhole_flow 1290440635
```

Example log is a icmpv6 packet send to an unused IPv6 address.

On the monitor host side, syslog-ng3 application is used for collecting logs and inserting them into database. syslog-ng3 is started with below configuration file. In the configuration file, destination is defined as a mysql database. syslog-ng3 gets logs from udp port 510, parses them and inserts them into mysql database. Database table is created automatically via syslog-ng3.

```
1  options { long_hostnames(off); flush_lines(0); };
2
3  source net {
4      udp6(port(510));
5  };
6
7  parser p_kovan {
8      csv-parser(columns( "KOVAN.SRC_IP", "KOVAN.SRC_PORT",
9      "KOVAN.DEST_IP", "KOVAN.DEST_PORT", "KOVAN.PROTO",
10     "KOVAN.MSG", "KOVAN.TIME")
11     flags(escape-double-char,strip-whitespace)
12     delimiters(" ")
13     quote-pairs('""[]')
14     );
15 };
```

```
16
17 destination d_mysql {
18     sql(type(mysql)
19         host("localhost") username("logger") password("231905")
20         database("kovan")
21         table("logs")
22         columns( "logtime INT UNSIGNED","service varchar(32)",
23         "srcip varchar(64)", "srcport int", "destip varchar(64)",
24         "destport int", "proto ENUM('tcp', 'udp', 'icmp6', 'icmp')",
25         "msg varchar(256)" )
26         values( "${KOVAN.TIME}", "$PROGRAM", "${KOVAN.SRC_IP}",
27         "${KOVAN.SRC_PORT}", "${KOVAN.DEST_IP}", "${KOVAN.DEST_PORT}",
28         "${KOVAN.PROTO}", "${KOVAN.MSG}" )
29         indexes("service")
30     );
31 };
32
33 log { source(net); parser(p_kovan); destination(d_mysql); };
34
```