# RATSPIT
# Remote Automatic TCP Stack Probing and Inspecting Tool
# CS261 Final Report
# Fall 1998

Joseph Gebis
gebis@cs.berkeley.edu

December 13, 1998

***TCP stacks respond differently in some situations where the TCP specification does not fully define the appropriate behavior. The exact behavior of the stack in these situations can be determined remotely by examining the responses to specifically-designed packets. I describe a tool that remotely probes machines to determine characteristics about their TCP stack. The method of probing the remote host as well as motivation for determining this information is discussed. I also describe further probing of network stacks, including probing IP-specific functions.***

## 1 Introduction

The Transmission Control Protocol specification [rfc793] defines the behavior of tcp implementations. It covers all the normal operations (such as opening and closing connections, establishing connections, and so on), as well as common error or unexpected communications (such as receiving a RST ("reset the connection") control packet before a connection has been established). The specification also covers many other aspects of communications that will not concern us.

The specification does not, however, cover all possible situations. There are combinations of states and control packets, as well as types of control packets, for which the appropriate behavior is not covered in the specification.

Virtually all TCP implementations correctly operate under normal conditions (or else they would not be able to communicate at all). Some implementations do not correctly implement some aspects of the protocol that are specified, but are not often excercised. Implementations vary in the way that they handle situations that are not fully specified.

Being able to determine the behavior of TCP implementations can be useful for many reasons. By doing this, one is able to check to see if the implementation matches the specification. Determining the behavior also enables one to match that behavior (either in a new implementation, or to emulate the behavior of the implementation). By comparing the differences in different implementations, it is also possible to identify the implementation. [queso]

ratspit (Remote Automatic TCP Stack Probing and Inspecting Tool) is a tool that is able to test the behavior of TCP implementations in these situations.

The next section of this paper describes related work. Section three expands on the motivation for creating a tool like ratspit. The operation of ratspit is described in section four. The final three sections include the current state of the tool, future plans for ratspit, and a summary of the paper.

## 2 Related Work

There are two main classes of related work: similar tools (of which only one is known to exist by the author), and tools that complement ratspit.

The only similar tool is QueSO. [queso] QueSO is another tool that probes remote TCP stacks to determine behavior, but it does have some significant differences. It probes the remote machine using seven different packet types. These packets are similar to the packets that ratspit uses to probe: they are control packets with combinations of control codes that would not normally be used. QueSO does not attempt to fully map out the TCP state diagram; instead, it uses the responses to generate what amounts to a fingerprint of the TCP implementation. It then compares this to a set of previously-generated fingerprints. If it finds a match, it is able to determine the operating system on the remote machine.

The tools that complement ratspit most are intrusion detection systems (ID systems). ID systems attempt to passively monitor network traffic to determine if an attempt at an intrusion occurs. To do this, they must be able to accurately model the result of traffic on the machines being watched. There are difficulties in doing this with normal traffic – it becomes much more troublesome to model the behavior of remote systems if the traffic includes abnormal packets (i.e., any packet for which the resulting behavior is not fully defined by the TCP specification). By using a tool that probes remote machines, this problem can be overcome in two ways:

- by determining the remote OS type, then using knowledge of that OS behavior to model the TCP stack on the remote machine.
- by probing the behavior of the TCP stack directly.

The problem of ID systems dealing with abnormal packets is well-known. If the only source of these packets was buggy programs, they might be relatively rare, and the problem might not be large. Unfortunately, since correct behavior is difficult to get correct, attackers purposefully use packets of this type. [PtNe] contains a good discussion of the problems in dealing with this. It also contains an examination of what different ID systems do with different types of packets, and a description of what types of packets are accepted or rejected by various operating systems.

## 3 Motivation

There are four main reasons for using a system like ratspit:

- **Identifying bugs.** By mapping out TCP implemenation behavior, it is possible to compare the actual behavior against the bahavior in the TCP specification. This can aid in implementing a TCP stack that operates correctly.

- **Identifying OS.** QueSO already uses abnormal packets to probe TCP stacks to determine their type. This can be useful to see there are computers in an organization that need to be upgraded, or simply to get a survey of what operating system are being used.

- **Matching behavior.** If one wants to implement a TCP stack that matches the behavior of an existing stack, it is not enough to simply implement the behavior in the specification (even if we ignore the possibility of bugs in the previous implementation). A tool like ratspit will show how the existing implementation reacts in each situation, and then this behavior can be duplicated.

- **Intrusion detection systems.** As mentioned above, ID systems try to emulate existing TCP stacks. Most systems now simply have some set behavior for reacting to abnormal packets (e.g., always accepting or rejecting packets of a certain type). With a system like ratspit, ID systems can either determine which operating system is in use, or simply determine how the remote machine reacts to packets of any type.

# 4 Operation

## 4.1 TCP variation

The TCP specification defines a state diagram that is shown in figure 1. This state diagram shows the normal actions that occur on most connections. For example, a server would start in the LISTEN state; upon receiving a packet with the SYN flag, the server should send a response with the SYN and ACK flags, and move to the SYN RCVD state. If the server then receives an ACK of the SYN, the server should move to the ESTAB state. These state transitions correspond to the server side of a normal three-way handshake.

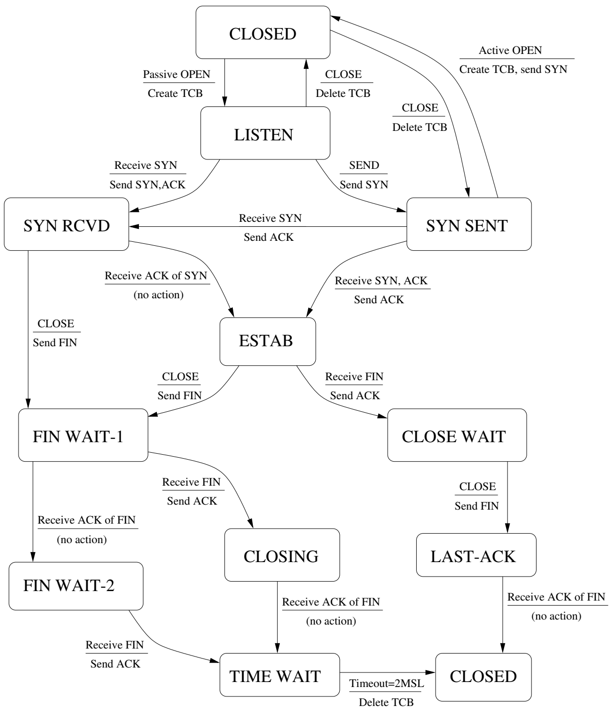


Figure 1: TCP state diagram

There are some possibilities that are not included in the state diagram. For example, what is the correct behavior if a TCP stack is in the SYN RCVD state, and it receives a packet with the FIN flag? One possibility is to move directly to the CLOSE WAIT state, as if you had received an unsolicited FIN to a set up connection (i.e., a connection in the ESTAB state). Another possibility is to send a response with the RST flag, and tear down the connection.

In many cases, the TCP specification is not clear. It says:

> As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case.

Does this apply to the previous example? The definition of a FIN flag, again from the TCP specification, is:

> A control bit (finis) occupying one sequence number, which indicates that the sender will send no more data or control occupying sequence space.

Since it does not make sense to send this before a connection has been established, it seems reasonable to assume that this packet was not intended for the current connection; then, a RST should be sent. On the other hand, it's not completely clear that this was the case: maybe the remote TCP stack knew that it wasn't going to send any data or control in sequence space, so it decided to send a FIN flag; then, the specification says that a RST must not be sent.

Since there is no clear single correct way to implement this behavior, different TCP stacks operate differently. There are many other cases where this can occur: almost any place in figure 1 where there is a state that does not completely define the correct behavior in the case of different control flags (and combinations of control flags) there is the possibility for differing behavior in different implementations.

## 4.2 Mapping the state diagram

The full behavior of a particular TCP stack can be defined by mapping the state diagram for that stack. Mapping the state diagram involves moving the TCP stack to a particular state, and excercising all possible combinations of control options.

Moving to a particular state is not particularly difficult for most of the states in the TCP diagram. For a remote server, a program such as ratspit will be able to move the server to:

- **LISTEN** This is the state the server starts in.
- **SYN RCVD** By sending a SYN from the LISTEN state.
- **ESTAB** By ACKing the received SYN from the SYN RCVD state.
- **CLOSE WAIT** By sending a FIN from the ESTAB state.

The remaining states may or may not be able to be reliably remotely probed by a tool such a ratspit:

- **SYN SENT** This state requires the connection to be iniated by the remote host. This may be possible to achieve, depending on how the remote host is set up: for example, an ftp connection can be made to the host (if anonymous ftp is allowed), and a LIST command issued. This will request that a file listing will be sent. This satisfies our requirements, because the connection for sending the file listing is not the same as the control connection used to start the ftp connection; it is a separate connection that is opened by the server. Thus, the server may be able to be put into a SYN SENT state.

- **FIN WAIT-1** This state can is shown to be the result of actions from two different states in the TCP state diagram. If a local CLOSE is issued from either the SYN RCVD or ESTAB state, the TCP stack will move into this state. Again, this state can be possible to achieve: it requires using some protocol where the server has a finite amount of data to send. An example of this is again ftp. Once the LIST command is issued, a file listing will be sent. After this list is complete, the server will close the connection and move into the CLOSE state.
- **FIN WAIT-2, CLOSING, TIME WAIT** These three state depend upon being able to move the remote stack into the FIN WAIT-1 state. If this can be achieved, then the stack can be moved into the TIME WAIT state through the CLOSING state by sending a FIN before ACKing the remote host's FIN; the stack can be moved into the TIME WAIT state through the FIN WAIT-2 state by ACKing the remote host's FIN before sending a FIN.

Even in the states that should be able to be probed on any server, ratspit has to produce the packets that move the remote stack to the appropriate state, and then produce abnormal packets to test the behavior of that state.

### 4.3 ratspit network interaction

Figure 2 shows the necesarry network interaction for ratspit to probe and analyze remote machines. It shows the flow of traffic in two directions: from the host on which ratspit is running (showing traffic from ratspit itself), and to the host on which ratspit is running (showing both traffic destined for ratspit, as well as normal traffic destined for other processes on the host on which ratspit is running).

### 4.4 Generating packets

ratspit must be able to generate packets that would not normally be generated by the host (i.e., packets with abnormal combinations of control flags, or control packets generated at unusual times). To do this, it must interact directly with the network. ratspit therefore employs raw sockets. Furthermore, to have complete control over the packets, ratspit uses the IP_HDRINCL option that allows it to specify complete IP packets, including headers. An example of the setup code is shown in Figure 3.

The initial `socket` call options (AF_INET, SOCK_RAW, and IPPROTO_RAW) specify a raw socket. Once this socket is created and the IP_HDRINCL options is set, raw packets can be sent by calling `sendto` with a full raw packet as the message.

### 4.5 Listening to packets

Besides generating the appropriate packets to probe a remote host, ratspit also has to be able to listen to and interpret the responses. There are two parts to listening to reply packets: making sure that ratspit receives the packets it needs, and making sure the host operating system does not interpret responses to ratspit's packets.

It is important that the host operating system does not interpret ratspit's packets. Because ratspit sets up and tears down its own connections, the host OS does not have any knowledge of the current state of interactions. Responses which

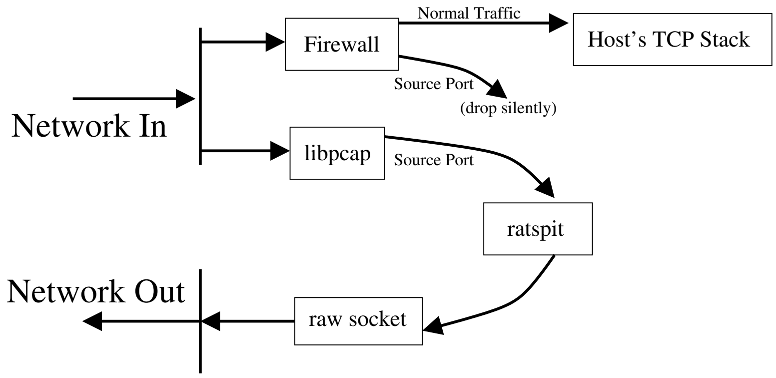


Figure 2: ratspit network interaction

```
int sock, retval;
char on = 1;

/* Create the socket */
sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if(sock == -1)
return -1;
/* Include headers */
retval = setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));
```

Figure 3: Example code for setting up raw socket

are meaningful to ratspit (including those that are simply used to move the remote host to the correct state to probe) would appear to the host machine as packets that are not destined for any active connection, and RST responses would be generated for those. The remote machine would interpret the RST as it should, by tearing down the connection.

It is possible to tell the host machine that it should not interpret packets destined for a specific port on the local machine. This is accomplished by firewalling the specific local port that ratspit uses. An example of how to do this using the Linux ipfwadm tool [ipfwadm] is:

```
ipfwadm -I -i deny -P tcp -D
0.0.0.0/0 5000
```

The "`-I`" option specifies the rule as an input rule; the "`-i`" option inserts the new rule at the beginning of the policy list; "`-P tcp`" limits the rule to TCP packets; "`-D 0.0.0.0/0 5000`" specifies the destination host and port, and corresponds to any destination host address and port 5000.

Firewalling the local port prevents the OS from interpreting packets that are destined for ratspit, but ratspit itself still needs to be able to get access to those packets. This is accomplished by using the libpcap library. [libpcap] This library is also used by the tcpdump tool [tcpdump] to accomplish a task similar to what ratspit is trying to accomplish: access to raw packets.

libpcap provides a simple interface to get access to packets of a certain type. Figure 4 shows an example of the setup required (although the example has a constant address coded in to probe for simplicity).

The code first looks up the default network device, and then opens it in promiscuous mode. After that, it compiles the filter program and sets the filter for the device. The only remaining work to be done is to call the procedure that sets a handler to be called for all incoming packets that pass the filter.

## 5 Current State

ratspit is not yet in a fully completed form. The tool right now is able to set up the network interaction that it needs to perform its tasks, and generate appropriate packets of the correct type. The code is available for download at `http://www.cs.berkeley.edu/~gebis/Projects/ratspit/`.

## 6 Future Plans

The first things that I am planning on working on are automating state mapping. Initially, I plan to get just the simple states (in section 4.2) mapped completely; even this will allow for a useful tool. I also plan on adding in the additional states (again, from section 4.2), although I am not convinced that using FTP to reach these states is the most effective and simple method of excercising those states.

The ratspit network interaction code would be relatively easy to separate from the rest of the package, with one exception: the firewalling code. It seems like a bad idea to try to integrate that more. Perhaps the best way to add this would be to use a perl script to install and run ratspit that could check which firewall rules are installed, and add any additional necessary rules.

It would also be nice to have the format be something easily readable. For example, a true, complete state diagram (that includes the behavior for all control flags and combinations of control flags in every state) would be a very useful addition.

Eventually, it would be nice also to include other sort of network probing in ratspit. However, I have no plans to do that at this time.

## 7 Summary

The TCP specification does not completely define appropriate behavior for some unusual circumstances. Different TCP stacks choose to implement different behavior in these situations. Discovering their exact behavior can be useful to find bugs, identify the remote operating system, match behavior of an existing implementation, and aid intrusion detection systems.

ratspit is a tool that automatically determines the behavior of remote TCP stacks. It does this by probing the remote machine with packets containing unusual control flags and interpreting the results.

```
char filter_string[] = "host 192.168.4.1 and dst port 5000\0";
char error[PCAP_ERRBUF_SIZE];
char *dev;
pcap_t *p;
struct bpf_program fp;
int snaplen=1500, promisc=1, timeout=0;
char optimize=0;
bpf_u_int32 netmask=0xffffffff;

if((dev = pcal_lookupdev(error)) == NULL)
return -1;
if((p = pcap_open_live(dev, snaplen, promisc, timeout, error)) == NULL)
return -1;
if(pcap_compile(p, &fp, filter_string, optimize, netmask) == -1)
return -1;
if(pcap_setfilter(p, &fp) == -1)
return -1:
```

Figure 4: Example code for using libpcap

## References


[rfc791] Postel, Jon (ed.). *Internet Protocol – DARPA Internet Program Protocol Specification*, RFC 791, USC/Information Sciences Institute, September 1981.

[rfc793] Postel, Jon (ed.). *Transmission Control Protocol – DARPA Internet Program Protocol Specification*, RFC 793, USC/Information Sciences Institute, September 1981.

[Com] Comer, Douglas E. *Internetworking with TCP/IP, Volume 1.*

[CoSt] Comer, Douglas E. and Stevens, David L. *Internetworking with TCP/IP, Volume 2.*

[Tan] Tanenbaum, Andrew S. *Computer Networks*, second edition.

[Ste] Stevens, W. Richard. *Unix Network Programming.*

[queso] savage@apostools.org. *QueSO.* `http://apostools.org/projectz/queso/`

[PtNe] Ptacek, Thomas H., and Newsham, Timothy N. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection.* `http://www.nai.com/products/security/advisory/papers/ids-html/doc001.asp`

[tcpdump] McCanne, Steve, Leres, Craig, and Jacobson, Van. *Tcpdump.* `ftp://ftp.ee.lbl.gov/tcpdump.tar.Z`

[libpcap] McCanne, Steve, Leres, Craig, and Jacobson, Van. *Libpcap*. `ftp://ftp.ee.lbl.gov/libpcap.tar.Z`

[ipfwadm] Vos, Jos, and Konijnenberg, Willy. *Linux firewall facilities for kernel-level packet screening.* `http://www.xos.nl/linux/ipfwadm/paper/`

[Gr] Grennan, Mark. *Firewalling and Proxy Server HOWTO.* `http://okcforum.org/~markg/Firewall-HOWTO.html`

[Al] Al-Herbish, Thamer. *Raw IP Networking FAQ.* `http://www.whitefang.com/rin/`