Global Information Assurance Certification Paper

Under the radar:
A look at three covert communications channels

James P. Goltz

GIAC security essentials (GSEC)
Practical assignment version 1.4b, option 1

January 23, 2003

Abstract

In 1998 and 1999, the development of semi-automated computer attack tools brought forth an innovation in information attack tools: programs designed to allow the attacker to communicate with them, to communicate amongst each other, and to report back to the attacker. In order to deter detection and to hide the tools' intent, the developers of these tools designed them to use *covert communications channels*: network communications channels meant to be hidden from normal view and/or to obscure the content of the covert communications.

This paper examines three tools that employ such channels, tools which serve as good examples of different kinds of covert communication:

- Back Orifice, a Windows-based remote-control trojan
- the Loki Project, a covert communications proof-of-concept library
- Trin00, a distributed denial-of-service tool

These three tools are used as examples because each employs a covert channel for communication between an attacker and an attack tool, or between two attack tools. Each example tool also demonstrates a different type of covert channel:

- Back Orifice relies on the use of obscure UDP ports.
- Loki uses a traffic type not normally used for communication (ICMP packets).
- Trin00 uses obscure TCP and UDP ports, and also relies on a multi-tier architecture with both attacker-to-tool and tool-to-tool channels.

For each example, this paper looks at how the covert channel or channels are designed, how they are used, and how such channels might be detected and blocked. The paper then shows how the means of detecting and blocking each specific channel can be generalized into a "principle" for network designs more secure against such covert communication.

Contents

1 Definitions

Attacker A person attempting to gain unauthorized access or produce results undesirable to the legitimate owners of computer systems and networks. Because many authors use the term “hacker” to mean a particular kind of programmer, regardless of good or ill intent, “attacker” is used instead in this paper to refer specifically to those attempting unauthorized access. Hackers may or may not have benign intent; attackers (as the term is used in this paper) are assumed to have hostile intentions.

Attack tools Programs, automated or not, that attackers use to attempt to gain access as described above. This paper often abbreviates this to “tools”; the full term “attack tools” will be used when necessary to distinguish programs used by attackers from programs used in more legitimate fashion.

Border router A router that connects a private network or networks to the Internet. All traffic traveling from the private network to the Internet (or another outside network) or vice versa must traverse this router, so it is a logical place to consider blocking traffic.

Firewall A system designed to prevent unauthorized access to or from a private network. [Webopedia.com, 2002] Often a firewall is more sophisticated in its traffic filtering than a border router: it can match outgoing DNS queries with their replies, for example, whereas a router may be limited solely to filtering based on source and destination address and port numbers.

Although a firewall need not be a router, and a border router need not be a firewall, because both are logical “choke points” for limiting outside traffic we will use both of these terms to indicate places on a network where attacker traffic may be blocked.

Covert channel Not openly practiced, avowed, engaged in, accumulated, or shown [Dictionary.com, 2002]. In this paper, “covert” implies not only that the channel is “not openly practiced” but where the designer of the channel makes an active attempt to hide the channel from view.

For example, the Loki project code (section 3) puts communications data into unused portions of ICMP packets, where no data are normally found. Postfix [Venema, 2002], on the other hand, prepends a “Delivered-To” header line[1] in the header of email, where diagnostic and control messages are routinely placed. Even though the “Delivered-To” header is normally hidden from view by a user’s mail reader program, it is not deliberately hidden. mail reader programs.

Trojan Horse A program designed to look innocuous but act dangerously. A typical Trojan Horse (“trojan for short”) will appear to be a game or useful utility in order to

[1] Prepended to email upon delivery to prevent mail routing loops.

entice an unsuspecting user into running it; but once executed, it will secretly install its payload (for example, the Back Orifice server).

The original Trojan Horse was a large, hollow wooden horse built by Greek soldiers during the Trojan War. The soldiers hid inside the horse, and when the Trojan soldiers brought the horse within the walls of Troy, the Greeks emerged and slaughtered the Trojans. [Canary, 2002]

2 Back Orifice

2.1 History

On August 1, 1998, a hacker group called "the Cult of the Dead Cow"[2] released Back Orifice. According to the press release, Back Orifice was designed to allow remote control and monitoring of Windows computers, ostensibly for legitimate systems administration purposes:

> [. . .]Back Orifice is a self-contained, self-installing utility that allows the user to control and monitor computers running the Windows operating system over a network.
>
> Sir Dystic [the programmer who developed Back Orifice] sounded like an overworked sysadmin when he said, "The two main legitimate purposes for BO are, remote tech support aid and employee monitoring and administering [of a Windows network]." [Cult of the Dead Cow, 1998]

2.2 How the tool is used

Back Orifice and its successor, Back Orifice 2000 (abbreviated "BO2K"), are intended to allow an attacker to control an infected host remotely. The program allows him to do almost anything that he could do while sitting at the keyboard of the host.[3] The program is presented as an tool for remote administration, but goes to some lengths to hide itself from view and escape detection. Its main use of late is as a payload of a trojan-horse program: the trojan infiltrates the Windows system and installs Back Orifice for later use.

2.3 Method of covert communication

(The factual information for this section is from Flávio Veloso's paper "The Back Orifice (BO) Protocol" [Veloso, 2001].)

[2] http://www.cultdeadcow.com/

[3] The name "Back Orifice" is a play on words of "Back Office", a Microsoft program suite that allowed, among other things, remote server administration. Back Office was less surreptitious than Back Orifice, however.

The base method of communication with a Back Orifice server is via UDP port 31337 [Internet Security Systems, 2002], although this can be changed in the source code (for Back Orifice) or a configuration file (for BO2K).[4]

Back Orifice packets have a 17-byte header containing an eight-byte "magic number" ("`*!*QWTY?`"), a four-byte integer containing the packet length (including the header and a one-byte CRC), a four-byte integer packet identification field, and a one-byte field containing flags and operation type (Figure 1).

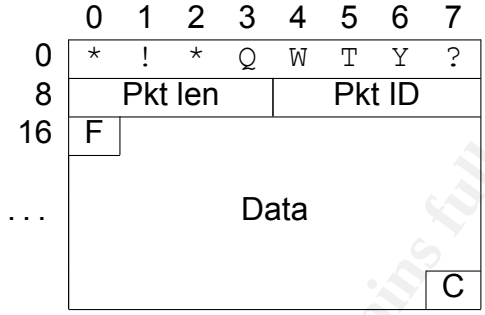

Figure 1: Layout of Back Orifice packets (F = flags/opcode, C = CRC)

BO packets are encrypted by XORing each byte with the output of a random number generator, which is initialized with a shared secret password. Because the first eight bytes are fixed and therefore always known, the encryption is relatively weak and can be broken. Various Back Orifice plugins exist that can more strongly encrypt the packets, however.

2.4 Detecting the covert channel

If the packet can be decrypted, detection is simple. Unmodified BO code will send packets to and from UDP port 31337, and these packets will have "`*!*QWTY?`" as the first eight bytes of the (decrypted) data portion. Obviously things aren't always that simple; it's relatively easy to modify the source code or configuration file so that Back Orifice uses a different port or different encryption. Even so, however, bytes 8 through 11 of the data portion will contain the length of the data portion (including header and CRC), so any packet that fits this pattern may be a BO packet. The chances of truly random data matching this pattern are approximately one in 4.2 billion[5]. Moreover, we can disregard any packet that, if it were a BO packet, has a "length" field less than 19, since a BO packet cannot be shorter than 19 bytes (17 byte header + 1 data byte + 1 CRC byte).

Packet data is not always random, however, and in practice false positives are much more common than one in 4.2 billion.[6] A protocol designed in a similar fashion to the

[4] If you view 3 as "E", 1 as "I", and 7 as "L", 31337 spells "ELEET" (elite). This made 31337 an obvious port for attacker use, which is why Back Orifice allows the attacker to change it.

[5] An unsigned 32-bit integer can range from zero to 4,294,967,296. Obviously, for a packet of a given length only one of these numbers matches the packet length.

[6] Based on personal observations of the NFR IDA by the author while he worked at NFR Security, from 1999 through 2001.

BO protocol, such as that used by servers for the Unreal Tournament online role-playing game, would trigger a large number of false positives.[7]

A better approach would be to count "BO-like" packets per host and see if one host (or a small number of hosts) are responsible for the majority of those packets. Detection efforts could then focus on those hosts.

2.5 Blocking communications

Because of the high probability of false positives, blocking every packet that looks like a BO packet is not feasible, as it would likely result in a large amount of legitimate traffic being discarded. Selective blocking of BO-type packets from hosts that may be infected (as detected by the methods in section 2.4 above) might yield better results.

2.6 Generalization

It is difficult to generalize these methods of detection and blocking, as they focus on specifics of the BO protocol. We see how knowledge of the protocol helped our efforts, however, so we can reasonably assume that knowing some specifics of the protocol we are attempting to detect and/or block will help us do so. Conversely, knowing the specifics of protocols normally allowed on your network and examining packets that fall outside of this allowed set would be an alternate (although more complex) approach.

3 The Loki Project

3.1 History

Loki was introduced to the public in *Phrack*, an online hacker magazine, in August and September of 1996 [Daemon9, 1997][Daemon9, 1996]. *Phrack* volume 49 contained an article on the basic theory; volume 51 contained an example implementation.

3.2 How the tool is used

Loki by itself is not an attack tool. Instead, it is essentially proof-of-concept code meant to be incorporated into other tools. Unlike the author of Back Orifice, the author of Loki makes no attempt to legitimize his work. He flatly states the covert nature of the protocol, as in this excerpt from *Phrack 49*:

> Loki is not a compromise tool. It has many uses, none of which are breaking into a machine. It can be used as a backdoor into a system by providing a covert method of getting commands executed on a target machine. It can be used as a way of clandestinely leeching information off of a machine. *It can be used as a covert method of user-machine or user-user communication*

[7] Observations by the author at NFR Security.

> [emphasis added]. In essence the channel is simply a way to secretly shuffle data (confidentiality and authenticity can be added by way of cryptography). [Daemon9, 1996, file 06]

In essence, Loki is proof-of-concept code, the concept being that of moving data over a covert channel by means of ICMP or DNS packets.

3.3 Method of covert communication

Loki uses ICMP Echo and Echo Reply packets as the basis for its covert channel. These packets are normally used to confirm that a particular host or router is "up" and processing network traffic. Because of the diagnostic nature of ICMP traffic, including Echo and Echo Reply packets, ICMP is often allowed unhindered through border routers and firewalls. Even firewalls that attempt to match up Echo and Echo Reply packets may be fooled by Loki, since a command in an Echo packet results in an acknowledgment in an Echo Reply packet.

RFC 792 [Postel, 1981b] states that ICMP Echo and Echo Reply packets contain a "data" portion, but defines no use for this portion other than to state "The data received in the echo message must be returned in the echo reply message." Loki hides its messages in this data portion.

Loki can encrypt its data using either a simple exclusive-OR (XOR) method or a more sophisticated Diffie-Hellman key exchange protocol [Levy, 2003][Diffie and Hellman, 1976] to synchronize on a Blowfish encryption key [Schneier, 1994]. This encryption does not affect the ICMP sequence number or the opcode (command) byte in the data portion.

A later version of the Loki protocol used UDP packets masquerading as DNS queries and query answers as a covert channel. The messages were hidden in the portion of the packet immediately following the UDP header. Many packet filters are configured to allow DNS traffic to pass in order to allow DNS lookups, so it is likely that such Loki packets could penetrate into an inside network.

Although the packet is UDP and either originates from or is sent to port 53 (the standard DNS port), there is no apparent attempt to make the packet data look like a valid DNS query. The original source code did not provide for encryption when using UDP transport, although this could easily be added.

Loki identifies packets as Loki packets by two basic means:

- The ICMP packet sequence number must be hexadecimal `0xf001` (possibly readable as the word "fool").
- Byte 0 of the data portion must be one of a limited number of "opcodes" indicating encryption keys to be used, request/reply, acknowledgment, and so forth.

3.4 Detecting the covert channel

Theoretically, detecting ICMP-based Loki communications is extremely difficult. Since ICMP Echo and Echo Reply packets may contain arbitrary data payloads for purposes

of link testing, there is no foolproof way to tell whether, for example, a given ICMP Echo packet is part of a Loki covert channel or simply an Echo packet with a random data payload.

In practice, however, three characteristics of Loki packets make them stand out:

- The sequence number is always `0xfoo1`. Even if this is changed in the source code as the Back Orifice port was (see section 2.3), we could simply do a histogram analysis of ICMP packet sequence numbers and look for "spikes" in the data. Since the sequence number is a 16-bit unsigned integer, we would expect to see a given sequence number no more than once every 65,536 ICMP Echo packets on average. More than three or so packets with the same sequence number, especially to or from the same host, would be cause for suspicion.

- The first byte of the data payload is an opcode, and therefore falls within a relatively small set of valid values. A statistically large number of ICMP packets with these values as byte 0 of the data payload would indicate packets worth looking at in more detail.

  It's worth noting, however, that although the Loki code as given complains about packets with an invalid opcode, it is possible that an attacker could change the code to silently ignore bad opcodes and then inject many Loki packets with invalid opcodes into the packet stream in order to foil this analysis. It would take a great many such packets to obscure the small number of valid packets, however, thus greatly decreasing the effective bandwidth of the protocol.

- The packets all come from or are sent to a given host or set of hosts. Statistical analysis of source and destination hosts for ICMP Echo and Echo Reply packets (similar to the opcode analysis above) would show a small number of hosts being responsible for the majority of the traffic.[8]

3.5 Blocking communications

The author of the Loki articles says about blocking the Loki protocol:

> Disruption of this channel is simply preventative. Disallow `ICMP_ECHO` traffic entirely. `ICMP_ECHO` traffic, when weighed against the security liabilities it imposes, is simply not *that* necessary. [Daemon9, 1996, file 06]

A less severe measure than disallowing all ICMP Echo traffic on your networks, is to disallow ICMP Echo traffic crossing your firewall or border router. Most of the hops an IP packet takes are outside of both your network and that of the person trying to connect to your servers; allowing "pings" from his border router to yours will detect these problems

[8]We would need to ignore ICMP Echo packets from known network monitoring hosts, since these hosts would be expected to generate many such packets in order to monitor the network.

without compromising security inside your network. Problems that lie within your network can be diagnosed by using Echo packets that do not cross your border router.

In addition, detection via statistical analysis as laid out in section 3.4 can be combined with reactive blocking of ICMP traffic to or from suspiciously “chatty” hosts until they can be scanned and verified to be “clean”.

Loki traffic over UDP to port 53 is even easier. ICMP Echo packets might conceivably be sent to any given IP device. In contrast, there is no reason to send a DNS query or query response to anything but a DNS server. Indeed, since client programs are usually allocated a port at random, there should be virtually no UDP packets originating from port 52 unless their originating host is a DNS server.

3.6 Generalization

As with Back Orifice in section 2.6, knowledge of the protocol is important in detecting and blocking it. In addition, however, we see here the importance of allowing only necessary traffic across your network perimeter. In this case, we analyzed the need for ICMP Echo and Echo Reply packets to cross the perimeter, and decided that the need for security far outweighed any network diagnostic benefits.

In the case of blocking UDP Loki packets, we find another lesson: *know your servers*. Only certain dedicated servers should be originating UDP traffic from port 53; any other hosts doing so are cause for alarm.

4 Trin00

4.1 History

Trin00 (pronounced “trin-oo”) was one of the first *distributed denial-of-service* (DDOS) tools. Prior to its emergence in November 1999, a number of denial-of-service (DOS) attacks were known [CERT Coordination Center, 1996], intended to consume resources of the target (such as bandwidth or computing power) and thereby making it impossible for the target to provides its usual services (such as email or web service). These attacks were individual in nature, however: an attacker would have one host attack another host, one at a time.

The emergence of Trin00, and later tools such as Tribe Flood Network[Dittrich, 1999b], TFN2K [Barlow and Thrower, 2000], Stacheldraht[9] [Dittrich, 1999a], and mstream [Dittrich, 2000], made denial-of-service attacks much more devastating by introducing a level of automation to the process. Instead of an attacker initiating denial-of-service attacks manually, he would command one or more *master hosts* to perform the attack. These masters would then each contact several *daemons*, hosts which had been compromised earlier and were listening for such instructions. These daemons would carry out the actual denial-of-service attack, but since one master could control hundreds or even thousands

[9] German for “barbed wire”.

of daemons, the attack could be much more devastating. In one such attack, servers belonging to Yahoo, eBay, E-Trade, Buy.com, and CNN were made virtually unavailable by floods of traffic that consumed the bandwidth of T-3 and larger telecommunications lines [Harrison, 2000].[10]

Non-distributed denial-of-service attacks often involve such actions as "connecting" the UDP echo and chargen ports (ports 7 and 19, respectively, as noted in [Postel, 1981a]) together, so that the hosts involved use all of their available resources generating and echoing characters over the network. By contrast, distributed denial-of-service tools seek to consume resources through sheer weight of numbers, by creating so much network traffic to a small number of hosts that either legitimate traffic is crowded out or the targeted hosts simply cannot handle the legitimate traffic because of the flood of other traffic.

4.2 How the tool is used

Trin00 and similar distributed denial-of-service tools use a two-tiered architecture (Figure 2). An attacker selects a target or targets for the DDOS attack and sends the IP addresses of the targets, along with the necessary commands, to one or more masters, which in turn send commands to numerous daemons, which perform the actual attack. The master can also shut down the Trin00 process on the daemon (presumably to prevent detection), and the attacker can in turn shut down the Trin00 master process.

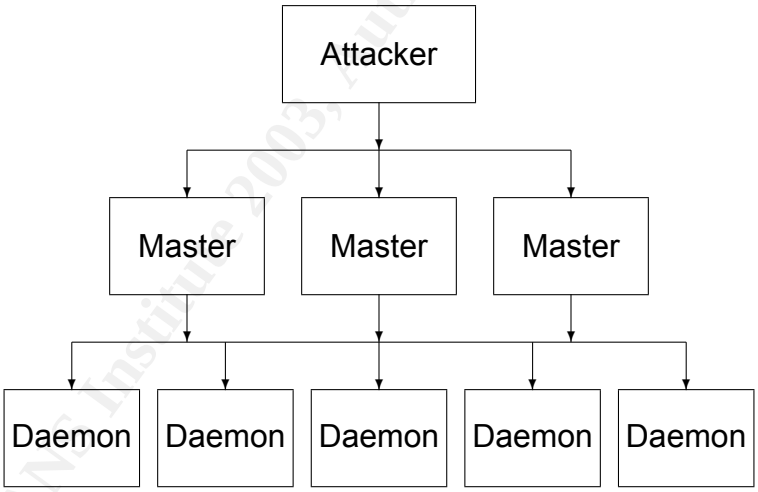

Figure 2: Generalized DDOS architecture

Trin00 daemon programs are installed on already-compromised hosts. The process is usually given a name identical or similar to existing network processes, so that process listings will not raise suspicions.[11] As part of the installation process, a crontab entry is

[10] A T-3 line carries approximately 45 megabits per second of network traffic.

[11] Alternately, a "root kit" may be installed on the system to hide the presence of the Trin00 daemon program. See [Miller, 2002][Dittrich, 2002].

created that restarts the daemon process every minute, presumably in case of crashes or sysadmin "interference".

Upon starting, each daemon sends the string "`*HELLO*`" to a precompiled list of master hosts on a covert channel. It can also send this string upon command, in order for the master to weed out non-responding daemons.

4.3 Method of covert communication

Trin00 has two covert channels: one for the attacker to communicate with the masters, and the other for the masters to communicate with the daemons. We will refer to these as the *master channel* and the *daemon channel*.

The master channel involves a TCP connection to port 27665. A password is required shortly after the connection is established ("betaalmostdone" [beta almost done] by default), and only one connection at a time is allowed.

The daemon channel uses UDP, with the daemon listening on port 27444 and the master on port 31335. Commands from the master to the daemons uses a password embedded in the command line, presumably because of the connectionless nature of UDP.

It is interesting to note that neither the master channel nor the daemon channel are encrypted in Trin00. Later distributed denial-of-service tools such as Stacheldraht use Blowfish encryption, making detection and analysis more difficult.

4.4 Detecting the covert channel

Like Back Orifice, Trin00 uses ports known to both sides of the channel for its communication. Traffic to or from an unaltered copy of Trin00 should be easy to spot on the network, since it will be using TCP ports 27665 and UDP ports 27444 and 31335, and sending unencrypted command traffic like "`aaa l44adsl` *`ip-addr`*" (DOS attack on the host at *ip-addr*) or "`mping`" (send a "`png`" command to all known daemons).

Obviously any of this can be modified in the source code: the ports used may be different, the passwords may have been changed, and even the command names may have been altered. Like any other network program, however, these programs cannot use a port already in use by another process, so if a well-known port is in use on a server that should not be providing that service, we should be suspicious. For example, if a given host is listening on UDP port 53 and that host is known not to be a DNS server, it may be that a Trin00 daemon is listening on that port instead. By knowing our network, and what services should and should not be running on any given host, we can spot "unusual" traffic fairly quickly.

In addition to the known TCP/UDP ports, Trin00 uses known file names for its daemon lists. The master program creates a file named "`...`" that contains the current list of known daemons.[12] This file is backed up to "`...-b`" if a new list is being generated as described in section 4.2.

[12] Unix systems normally do not display files whose name begins with "`.`" in directory listings. In addition,

This file may be encrypted using Blowfish, but its very presence should raise a red flag. Even if the source code is altered and the file is named something else, the sudden appearance of a new file or the inexplicable altering of an existing file on a carefully watched server should invite further investigation.

But what about a desktop system? These cannot be watched as closely as a server; often they are not centrally managed, and even if they are their user may create any number of new files one day — documents, spreadsheets, slide shows, and so on — and delete some or all of them the next. In this case we cannot rely on file system monitoring as noted above for servers; but on the other hand, desktop systems should not be providing services, so scanning for open ports might tip us off.

4.5 Blocking communications

Apart from detecting unusual activity in the file system, our best bet for blocking distributed denial-of-service covert channels is the fact that they must use known ports (known to the attacker who compiled and installed them, that is). Assuming we know our network, the attacker is at a disadvantage here:

- If he uses otherwise unused ports such as 27444 and 31335, his traffic should be blocked at our border router. Since we are not providing any services that use these ports, we have no need to allow traffic on these ports through our border router.
- If he uses well-known ports such as TCP port 22 (SSH) or UDP port 53 (DNS) (hoping that our border router is not blocking traffic on these ports) and if we know which of our servers provide which services, we can limit traffic passing through the border router to only the servers that should get it. A compromised host may have a Trin00 daemon process listening on UDP port 53, but if it is not one of our known name servers we should not be letting traffic on port 53 through to it.

4.6 Generalization

Apart from the recurring mantras “know your network” and “know your systems”, what might be called the “Principle of Least Passage” applies here. Our main defense against unknowingly being part of a Trin00 or other distributed denial-of-service network is to allow through the perimeter only traffic that is part of a service we know we’re providing. In addition, we can see the advantage in doing network scans regularly to look for open ports on our systems, and host-based scans to look for odd changes on our systems.

5 Summary

We can summarize the “lessons learned” from the above examples in four maxims:

the current directory is always shown as “`.`” and the parent directory as “`..`”. Trin00’s authors were probably hoping that a file named “`...`” would often be overlooked.

Know your foe. Knowledge of the covert channel protocol — how data is encapsulated and what the protocol packets look like — gives us a great deal of information about how to detect and/or block the covert channel. (Admittedly, this often isn't possible with a brand-new kind of covert channel.)

Know your servers. Knowing what to expect from your servers and what services they provide helps detection immensely. If DNS queries appear to be answered by a system you know should not be a DNS server, you should be highly suspicious.

Additionally, scan your servers with a data integrity tool like Tripwire[13] or fcheck [14]. It's nearly impossible to know if a rootkit or other attack tool has been installed on a system if you don't keep track of what is installed on that system and what has changed recently.

Know your network. What services do you provide to the outside world, and from which servers? What protocols are allowed on your network, and what does their traffic look like? What protocols should *not* be seen on your network? It's much, much easier to spot traffic that doesn't belong on your network if you know what traffic *does* belong there.

Principle of least passage. What kind of traffic needs to cross from your inside network to your outside network, and vice versa? Consider setting your firewall or border router to pass this traffic and block everything else. If it doesn't need to get inside, don't let it.

6 Conclusion

In this paper we have looked at three tools which use covert communication channels. We examined the design and use of said channels, and examined ways to detect and/or block these channels. We also generalized the means of detecting and blocking the covert channels into four general principles for network design and maintenance:

- Know your foe (know the tools and covert protocols where possible)
- Know your servers (know what services are provided and scan servers regularly)
- Know your network (know what services and protocols traverse your network)
- Principle of least passage (only allow necessary traffic through border router)

While adherence to these principles will not completely secure a network (if anything can), they can server to make networks more secure against the use of these and other covert channels.

[13] `http://www.tripwire.com/`
[14] `http://www.geocities.com/fcheck2000/fcheck.html`

7 References

[Authority, 2002] Authority, Internet Assigned Numbers. IANA protocol numbers and assignment services directory — port numbers. Last updated 8 January 2003.
URL http://www.iana.org/assignments/port-numbers

[Barlow and Thrower, 2000] Barlow, Jason and Thrower, Woody. TFN2K — an analysis. 10 February 2000.
URL http://securityresponse.symantec.com/avcenter/security/Content/2000_02_10_a.html

[Canary, 2002] Canary, Bob. The Trojan war. 15 July 2002.
URL http://www.uwp.edu/academic/english/canary/trojans.html

[CERT Coordination Center, 1996] CERT Coordination Center. UDP port denial-of-service attack. Advisory CA-1996-01, CERT Coordination Center. 8 February 1996.
URL http://www.cert.org/advisories/CA-1996-01.html

[Cult of the Dead Cow, 1998] Cult of the Dead Cow. Back Orifice press release. 21 July 1998.
URL http://www.cultdeadcow.com/news/back_orifice.txt

[Daemon9, 1996] Daemon9. Project Loki: ICMP tunneling. *Phrack*, 7(49). 8 November 1996. Issues of *Phrack* are distributed electronically in one or more sequentially numbered files; the article cited is in file 06.
URL http://www.phrack.org/show.php?p=49&a=6

[Daemon9, 1997] Daemon9. LOKI2 (the implementation). *Phrack*, 7(51). 1 September 1997. Issues of *Phrack* are distributed electronically in one or more sequentially numbered files; the article cited is in file 06.
URL http://www.phrack.org/show.php?p=51&a=6

[Dictionary.com, 2002] Dictionary.com. Dictionary.com definition of “covert”. 2002.
URL http://dictionary.reference.com/search?q=covert

[Diffie and Hellman, 1976] Diffie, Whitfield and Hellman, Martin. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[Dittrich, 1999a] Dittrich, David. The “stacheldraht” distributed denial of service attack tool. 31 December 1999.
URL http://staff.washington.edu/dittrich/misc/stacheldraht.analysis

[Dittrich, 1999b] Dittrich, David. The "Tribe Flood Network" distributed denial of service attack tool. 21 October 1999.
URL http://staff.washington.edu/dittrich/misc/tfn.analysis

[Dittrich, 2000] Dittrich, David. The “mstream” distributed denial of service attack tool. 1 May 2000.
URL http://staff.washington.edu/dittrich/misc/mstream.analysis.txt

[Dittrich, 2002] Dittrich, David. “Root kits” and hiding files/directories/processes after a break-in. 5 January 2002.
URL http://staff.washington.edu/dittrich/misc/faqs/rootkits.faq

[Harrison, 2000] Harrison, Ann. Cyberassaults hit Buy.com, eBay, CNN and Amazon. *Computerworld*, 9 February 2000.
URL http://www.computerworld.com/news/2000/story/0%2c11280%2c43010%2c00.html

[Internet Security Systems, 2002] Internet Security Systems. advICE Security Database, Exploits/Ports/31337. Last modified 9 December 2002.
URL http://www.iss.net/security_center/advice/Exploits/Ports/31337/default.htm

[Levy, 2003] Levy, Benjamin. Diffie-Hellman method for key agreement. Last modified 30 October 2003. Based on [Diffie and Hellman, 1976].
URL http://www.apocalypse.org/pub/u/seven/diffie.html

[Miller, 2002] Miller, Toby. Analysis of the t0rn rootkit. 2002.
URL http://www.sans.org/y2k/t0rn.htm

[Postel, 1981a] Postel, J. RFC 790: Assigned numbers. September 1981. The information in this RFC is of historical value; the current port assignments are in an online database. See [Authority, 2002].
URL ftp://ftp.rfc-editor.org/in-notes/rfc790.txt

[Postel, 1981b] Postel, J. RFC 792: Internet control message protocol. September 1981.
URL ftp://ftp.rfc-editor.org/in-notes/rfc792.txt

[Schneier, 1994] Schneier, Bruce. Description of a new variable-length key, 64-bit block cipher (Blowfish). In *Fast Software Encryption, Cambridge Security Workshop Proceedings*, pages 191–204. Cambridge, U.K.: Springer-Verlag, Heidelberg, Germany, 1994.

[Veloso, 2001] Veloso, Flávio. The Back Orifice (BO) protocol. Last updated 22 November 2001.
URL http://www.magnux.com/~flaviovs/boproto.html

[Venema, 2002] Venema, Wietse. The Postfix official web site: Frequently asked questions: the Delivered-To header. File last modified 9 January 2002.
URL http://www.postfix.org/faq.html#delivered

[Webopedia.com, 2002] Webopedia.com. Definition of “firewall”. Last modified 24 September 2002.
URL http://www.webopedia.com/TERM/F/firewall.html