# Bypassing Firewalls: Tools and Techniques

Jake Hill <jah@alien.bt.co.uk>

March 23, 2000

**Abstract**

This paper highlights a very important problem with network perimeter firewalls. The threat discussed is not exactly new, but neither is it widely recognised—even amongst network security professionals.

Most commercial firewalls claim to be application layer devices, but they derive very little useful information about the context of the application traffic that passes through them. Malicious applications can misuse even the simplest protocols in a way that totally bypasses the firewall's controls. This paper describes the methods of simple protocol tunnels, and shows how they can be applied. It also considers ways to counter this threat, and suggests that architectures based on military security principles and IPSec can improve security dramatically.

**Keywords:** firewalls, protocol tunnelling, IPSec, security.

## 1 Introduction

Firewalls are regarded as the primary defence mechanisms when connecting private networks to the Internet. There are various types, but firewalls essentially control access at the application and transport layers, often using application layer information to make access control decisions. A firewall is there to stop unwanted traffic from the Internet entering the protected network. In a similar way, it can also control traffic flowing out on to Internet.

A firewall is a perimeter security mechanism. It has no control over what occurs on the internal network. It simply screens any connections made between the inside and the outside. The only information the firewall has about a particular connection, are the source and destination addresses and port numbers. This is not enough to reason about the information that is being communicated. Indeed, it is very easy to fool a firewall

into allowing a protocol that it is supposed to be blocking—either by changing the port numbers associated with that protocol, or by using a protocol tunnel.

The remainder of this paper introduces protocol tunnelling and shows how easily it can be used to bypass a firewall. It then goes on to consider how this threat can be countered. The reader will notice that this discussion does not suggest ways of strengthening the firewall. It focuses instead on in-depth mechanisms, which complement the perimeter firewall. These mechanisms are based on a combination of military-style trusted systems, and on IPSec network security.

### 1.1 Related Work

Independent work on protocol tunnelling was done almost simultaneously by Laars Brinkhoff [5]. The architecture described in section 3 builds on work by Landwehr et al. at Naval Research Laboratories [9], and by Dalton, Clark, and others, at HP Laboratories [1, 2, 3, 13].

## 2 Protocol Tunnelling

A protocol tunnel encapsulates one protocol inside another. Tunnelling is a general technique which can be used to carry a protocol across a foreign network. It is often used to join two isolated networks with a private *bridge* across a public network, forming a VPN (Virtual Private Network). Most commonly, IP traffic is encrypted and encapsulated in a TCP stream, which is carried across the public Internet between two remote sites.

Any protocol can be exploited for tunnelling. The only requirement is that the protocol is permitted by any firewall that sits between the tunnel end points. Protocols like SMTP and HTTP generally satisfy this requirement. Others, like ICMP-ECHO (the “ping” protocol), are also allowed by most configurations.

### 2.1 Bypassing Firewalls

A protocol tunnel can turn an application layer protocol (such as HTTP, or SMTP) into a transport layer protocol. This can make it very hard for the firewall to reason about the traffic passing through it.

One tool designed to exploit this fact is GNU httptunnel, which is available under GNU Public License. This tool creates a point-to-point HTTP tunnel, but it can be used

in conjunction with other software (such as SSH and Telnet) to provide unrestricted access through a firewall. Users at sites with restrictive firewall policies can enable protocols that are blocked by tunnelling them through HTTP.

The obvious problem caused by tools such as this is that the firewall policy no longer dictates the overall security policy. Although users must take a conscious decision to invoke applications like htc and hts[1], it is ultimately the users who can decide which particular protocols will cross the firewall.

## 2.2 Simple Tunnel

Far more nefarious scenarios are possible, such as one demonstrated recently at BT Laboratories. The Simple Tunnel[2] is another general purpose tunnel developed independently to GNU httptunnel, but at about the same time. It was built as part of a demonstrator, designed to highlight the threat that tunnelling poses to network security [6].

Like GNU httptunnel, the Simple Tunnel also uses HTTP as a transport, although it could easily be extended to almost any client-server protocol. However, it is packaged as a library and not as an application—it is designed to be built in to other applications. This library, called libtunnel, provides a channel for passing arbitrary messages between the tunnel endpoints. This messaging system can be used in higher level libraries and applications, for communicating with a remote host.

The operation of the Simple Tunnel is illustrated in fig. 1. The client and server queue messages at each end of the tunnel. The client makes periodic connections to the server. These connections are HTTP-like, in that the exchange follows the basic protocol of [7]. The algorithm used by the client is as follows;

1. If the client has messages to send, it makes an HTTP POST request to the server. The messages are encoded and sent in the body of the request[3]. Otherwise,

2. if the client has no messages to send, it makes an HTTP GET request to the server.

3. If the server has any messages to send, they are encoded and returned in the body of the response.

[1] These are GNU httptunnel's client and server applications.

[2] The name belies the fact that the software is actually rather complicated.

[3] The client could instead use HTTP GET and encode the messages in the URL of the request.

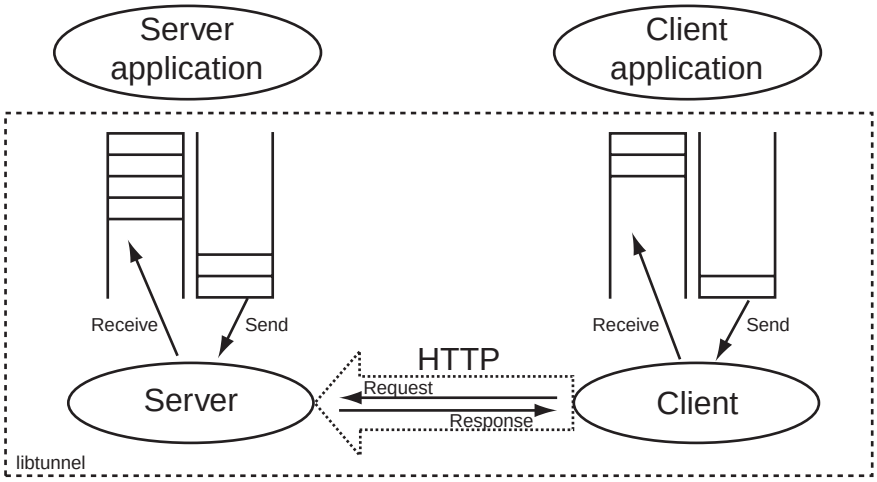


Figure 1: The Simple Tunnel library

The client can be configured to use a proxy, which allows it to work across an application-layer gateway. In addition, the server can manage tunnels to multiple clients simultaneously. The applications at each end of a tunnel use a simple API [12] to read and write messages, and also to control some of the tunnel characteristics.

The Simple Tunnel library has been used in some example applications. One of these is a remote socket library, called librsocket. It uses the messaging provided by libtunnel to tunnel system calls to the network stack of the client. The librsocket API [10] is very much like the standard socket API [11]. It was chosen because it is fairly simple, very useful, and implemented on a wide range on platforms. But almost any API on any machine can be tunnelled in the same way. The technique gives the attacker full access to the resources of a remote host.

While protocol tunnelling is a very powerful tool to use against firewalls, an attacker from outside must install client software on a machine behind the firewall. It is fortunate for the attacker that there are a great many ways this could be done, either by exploiting software vulnerabilities, or by social-engineering attacks. Some examples might be;

**Remote exploit.** A remotely exploitable bug in a program, which lets the attacker execute arbitrary code, may be used to bootstrap installation of a tunnel client.

**Trojan execution.** A user might be tricked into running a tunnel client, thinking it was something else.

**Viral infection.** A virus which can infect executable programs may contain a tunnel client. Even a macro virus might be made to carry a tunnel client, if the macro has access to networking functions.

This list of attacks is certainly not exhaustive, but it should illustrate that the job of installing malicious client software is not particularly hard. Within each category of attack, there are scores of real examples—the reader will find many of these documented in the archives of bugtraq[4] and CERT[5].

To demonstrate a Trojan Horse attack, librsocket was used to build a special version of the Mozilla browser. In the Macintosh version of this Trojan, this change required the addition of a single line of code to the Mozilla source, and relinking with librsocket. The resulting browser behaved in every way like a normal web client, except that it also made regular connections to the tunnel server. Whenever a user ran the Trojan browser, they unwittingly opened up the network stack of their machine for use by the "attacker".

# 3 Towards Defence in Depth

It has been said[6] that, the total security of a system is worse than the security of the weakest component in the system. This implies that the weak component in isolation is less of a problem than when it is integrated as part of a system, which depends upon the security of the weak component. It follows that, perimeter controls go only part way to providing security for a private network connecting to the Internet. A firewall can be bypassed if a host on the inside can be compromised.

As has been shown, a network of insecure systems can not have its security policy enforced by a firewall alone. But the problem is not the ability of the firewall, it is the security of the systems *behind* the firewall. A strategy of "defence in depth" is critical to preventing attacks such as the one just described. This term has gained popularity recently and it often used to refer to host-based detection of viruses or intrusions. But true in-depth defences should protect *against* these attacks too. This section considers the kind of "defence in depth" that is required to prevent a tunnelling attack.

[4] http://www.securityfocus.com/bugtraq

[5] http://www.cert.org/

[6] Perhaps by Bruce Schneier, perhaps by Carl Ellison.

## 3.1 Domains

Carroll and Landwehr [9] argue that the key characteristic of secure systems is their ability to maintain *domains* for storage and processing. A domain is a set of information, and *authorisations* for using that information. This notion stems from military security. It is not critical *how* domains are maintained, only that they can be prevented from interfering with each other, and from interacting with each other in ways that would violate the security policy.

The mechanisms to support "maintaining domains" can be implemented both in hardware and in software. Software controls can exist both at the system level and within applications. What *is* critical however, is that the controls work properly. This may seem obvious—but any flaws in low level mechanisms can completely undermine those at higher levels.

There are various examples of how these ideas can be applied to networked systems. Dalton and Clarke [2] have suggested how to allow secure access to LAN services from Internet hosts. Choo [1] shows how to extend this idea, by segregating the internal network in to VPNs running at different sensitivity levels. These, and other related schemes [3, 13], make extensive use of trusted systems like the CMW (Compartmented Mode Workstation).

A CMW adheres to the U.S. government criteria laid down in [4]. It it designed to enforce military security policy regarding the use of classified information. The CMW differs from a "conventional" workstation in several ways, including;

**Information labelling.** All information is stored in *segments* with an attached label. The label specifies the *sensitivity* of the information (e.g. SECRET, TOP SECRET) and also its *compartment* (e.g. NUCLEAR, COMMAND). The CMW ensures that the labels can not be altered without the correct authority, and that they are transmitted with the information across any network.

**Access control.** In addition to the normal concept of permissions, known as DAC (Discretionary Access Control), the CMW enforces MAC (Mandatory Access Control). This is the invariant security policy used by the military to control sensitive information. The MAC policy cannot be changed or overridden[7].

**Privilege and authority.** Rather than have a single "super-user", the CMW has a set of privileges and authorities which can be assigned to individual users and pro-

[7] It can be overridden, but only with the correct privilege.

grams. In a normal Unix system, common programs like ping and xterm need to run as root because they perform privileged operations. The ping program, for example, needs to open a raw network socket in order to send ICMP datagrams. On a CMW, ping only requires the privilege to open a raw socket.

Authority is a concept which gives different users the ability to run certain dangerous programs. Any user can run the ping program, but only the user with the right authority can mount or unmount filesystems. Authority allows administration of the system to be delegated to a number of different users, none of which have complete control of the system.

**Audit capability.** Logging provides a way of to audit the operation of the system. A CMW kernel logs all the system calls that processes make to it. This logging cannot be disabled. In addition, the CMW provides a way for applications to log their own actions in a secure way.

The object oriented Java programming language is another technology that supports "maintaining domains". In present day implementations, Java does this entirely within one application—the Java Virtual Machine (JVM). Because of this, the Java environment is vulnerable to faults in the underlying machine architecture. Java objects are prevented from interfering with one another but, very often, faults in applications that co-exist with the JVM can interfere with Java objects. Thus, security in Java still depends on trusted underlying systems.

## 3.2 Network Domains

A secure networked system needs to be able to maintain domains which span multiple hosts. Choo proposes an IPSec [8] solution for segmenting the internal network. He describes using CMW technology to enforce this segregation by running multiple IPSec stacks, at least one at each sensitivity level. In his scheme IPSec runs as a user-space process, which prevents stack errors affecting both other sensitivity levels, and also the kernel. Note though, that a single ISAKMP daemon is trusted by all IPSec processes. This does leave the whole scheme vulnerable to keying attacks, if the daemon can be subverted. ISAKMP is probably too complex to run as a trusted service, although it is difficult to see how else it could run in this scheme.

Coupled with this, the ideas in [1] do not consider the existing (insecure) systems that are attached to the internal network today. These systems are not manageable

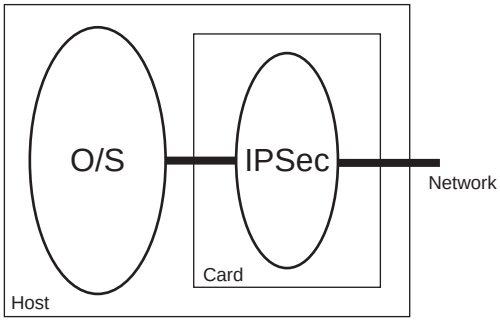


Figure 2: Bump in the wire

as part of a segregated network, so they would be limited to handling “unclassified” information. This would arguably make them useless in a corporate setting.

### 3.3 Bump in the Wire

The above ideas can be extended to include legacy systems too. The approach is to implement networking as a bump-in-the-wire for the untrusted hosts. The bump-in-the-wire replaces the inbuilt networking with a trusted plug-in version, implemented in firmware on a special network card. The card is in fact a minimal computer, running its own trusted network stack. Instead of a full network stack, the untrusted host has a set of simple stub functions which call on facilities of the card. This is illustrated in fig. 2.

All key material is maintained securely by the card, and it is never stored in the memory of the main computer. The system can preserve sensitivity labels across hosts much as suggested by Choo. The ISAKMP daemon establishes the sensitivity of the communicating socket endpoints, so labels do not need to be attached to packets in transit. The boundary conditions for the user and host are determined through the PKI. Hosts which are not capable of handling sensitive information are prevented from doing so by the network stack—by embedding it on a card, this can be assured to a high degree.

The bump-in-the-wire protects the operating system kernel from networking faults. The host itself is is still vulnerable to faults in the software running on it—the ability to maintain domains is still important. But the effects of faults in the software can be contained, through the policy of the bump-in-the-wire. This is effectively isolated from the rest of the machine. Even in kernel mode, the machine can only talk to the card through a very limited interface. It should be possible to configure and manage the card

entirely from the network side, from trusted hosts on the network.

# 4 Concluding Remarks

This work has demonstrated a particular weakness of all firewall designs. They can offer protection against certain types of network attack, but not against others. In particular a firewall cannot enforce an information security policy on the network it protects.

As information, and the flow of information, becomes increasingly critical to organisations, they will need to take additional measures to protect that information. These need to be defence-in-depth measures, which compliment the protection offered by the firewall. This work has highlighted technology which can provide these defences. Most of this technology is taken from military information security. The challenge now is to understand how it can be applied usefully in a business environment.

# References


[1] T. Choo. Vaulted vpn: Compartmented virtual private networks on trusted operating systems. Technical report, HP Laboratories, 1999.

[2] C. Dalton and D. Clarke. Secure partitioned access to local network resources over the internet. Technical report, HP Laboratories, 1998.

[3] C. Dalton and J. Griffin. Applying military grade security to the internet. Technical report, HP Laboratories, 1997.

[4] Compartmented mode workstation evaluation criteria, 1991.

[5] Gnu httptunnel. http://www.nocrew.org/software/httptunnel.html.

[6] Jake Hill. Using a protocol tunnel to subvert a firewall. Technical report, BT Laboratories, 1999.

[7] *RFC2616: Hypertext Transfer Protocol—HTTP/1.1.*

[8] *RFC2401: Security Architecture for the Internet Protocol.*

[9] C. E. Landwehr and J. M. Carroll. Hardware requirements for secure computer systems: A framework. In *Proc. 1984 IEEE Symposium on Security and Privacy*, apr 1984.

[10] *Manpage for rsocket(3N).* Distributed with libtunnel.

[11] *Manpage for socket(3N).* Available on most Unix-like systems.

[12] *Manpage for tunnel(3N).* Distributed with libtunnel.

[13] Q. Zhong. Providing secure environments for untrusted network applications: with case studies using virtualvault and trusted sendmail proxy. Technical report, HP Laboratories, 1997.