How to gain access to a secured website

written by Steinowitz

November 1999

Contents

1 Introduction

It happens so often: you're surfing across the Web and suddenly a box pops up. This box asks you to enter your user name and password for this particular part of the website. And if you don't know have these? Well, NO ACCESS, DONUT!

That can't be the intention of the Internet: shouldn't all information be freely available for everyone?

I'm not only writing about membership needed for *graphical information*, but also for many other types of information.

How do the protections of these secured websites work? And what can we do to gain access to secured websites? I'll try to answer both of these questions, although the first question will be more important. I won't give you specific details on all possible ways to gain access, but if you know how these protections work, it should be easier to eliminate them.

Please remember this document should serve as an introduction to the subject only. After reading this, you won't be able to access any secured site you want. You might be able to hack 1 or 2 sites, but you'll need to read much more if you want more.

2 What types of websites are secured?

There are numerous reasons which could lead to securing a website. Actually, there are so many reasons that I won't mention them all: I could write a whole document on that. In most cases, it has to do with a certain membership. I'll mention some possible cases.

- Free webbased email services - people are only allowed to see their own email.
- Company sites - some areas are only accessible for clients and/or employees.
- Commercial sites - accessible only for those who paid a lot of money for it.
- Non-commercial sites - even these sites may have member-only or otherwise private sections.

Reading this, I think we may conclude that websites in all categories we could think of could be secured for some reason.

3 Learning more about our target

It shouldn't be necessary to say that it's first necessary to learn more about our secured target website. There's no general approach which will enable you to access all secured websites. First, it's necessary to find out what type of protection we're dealing with. Since we're mainly talking about gaining access to secured websites here, it's obvious to start with our browser.

3.1 Using our browser

Fire up your browser (which should be Netscape Communicator 4.6 or higher) and surf to your target website: *http://www.target.com.* Naturally, we then try to get access to the secured section of the website which we would like to see. It's very important to analyse *how* users are authorized right now, because that will determine our approach to hacking it.

What kind of protection are we dealing with? Let me explain how you can recognize the possibilities. These possibilities can be divided in two types: server-side protections and client-side protections. Server-side protections can be rather difficult to tackle, client-side protections aren't much of a problem.

Webserver software I'll start with the best protection of all: server-side user authentication by the webserver. Very easy to recognize: a dialog box pops up which asks you to provide user name and password. Returns you a 'user authentication failed'-page if you choose 'Cancel'.

CGI You're dealing with a server-side CGI protection if there is a form in a 'normal' webpage which requires you to fill in some data, to click on a button and *if the data you provided is then transmitted to the server.* You can check this by looking at the page source: if there's a tag like <FORM ACTION="somename.cgi" METHOD="POST">, you know that you're dealing with a CGI protection.

Java applet I've seen sites protected with a Java applet quite regular, which is rather stupid, because Java applets are client-side. It's easy to recognize an applet: when you don't see it's a Java applet when browsing the website, you only have to look at the page source. The <APPLET SRC=appletname.class>-tag is the only way to load a Java applet in a webpage.

JavaScript Websites protected with JavaScript are the most stupid ones, because the user has access to all scripts used - it's also client-side. Also easy to recognize: view the page source and look at everything which is between <SCRIPT LANGUAGE="JavaScript">- and </SCRIPT>-tags. You'll see soon enough if it has something to do with the protection.

That are the four main protection types. You might be able to find one more, but these are definitely the ones used most.

3.2 Intermezzo: hostnames, IP addresses, protocols and ports

When you want to become a hacker, it's very important that you know everything about networks like the Internet. A couple of terms you should read about are: *hostnames*, *IP addresses*, *protocols* and *ports.* It may not always be necessary to know all about this, but it certainly makes life easier.

hostname Almost every computer connected to the Internet - or any other similar network - has a unique hostname, like this: *name.sub.domain.xx.* *xx* is the country code of the domain, *domain* is the domain name, *sub* is the subdomain name and *name* could be something like the computer name. Please note that not all hostnames necessarily match this format, however, a domain and country code are required.

IP address Every computer connected to the Internet - or any other similar network - has one unique IP address (some computers even have more). An IP address looks like this: *xxx.xxx.xxx.xxx*. Every *xxx* must be in the range 0-254 and the first *xxx* must be at least 1.

DNS DNS stands for Domain Name Service. It would be very annoying if we would all have to remember the IP addresses of all websites we would like to visit. Therefore, hostnames exist. When your Internet program tries to reach an Internet server, he connects to a DNS server and asks this server what the IP address of a certain hostname is. After that, using the IP address the DNS returned, your Internet program is able to connect to the server you originally wanted to connect to. It's also possible to 'reverse lookup': connect to a DNS server and ask what the hostname of a certain IP address is.

port All computers have a large number of virtual *ports*. When you want to connect to a remote computer, you also have to know which port you want to connect to. A program can *listen* to a certain port and accept a connection from any remote computer trying to connect to that port.

protocol A protocol is a set of rules which should make it possible for computers to communicate. Without such a protocol, a computer wouldn't know when it should receive and transmit data to the other computer. The HTTP protocol, for example, is used for transmitting webpages. When you surf over the Web using your browser, this protocol is used. The webserver software also uses the same protocol and therefore knows which files to send to the browser. (HTTP = HyperText Transfer Protocol)

Protocols and ports are related to eachother in most cases: when you tell your browser to surf to a website, it automatically connects to port 80, because that's the standard port for the HTTP protocol. And if you are lucky, the webserver software is listening to this port and answers your browser's requests.

(Note: it is possible to configure your webserver software in such a way that it listens to another port, 9000, for example. But this isn't very useful, since browsers automatically try to connect to port 80 and if your webserver software isn't listening to that port, no connection will be established. More about this later in this essay.)

I suggest you read more about these if you feel you don't know enough about them...

3.3 Finding our more using other protocols

In most cases, a webserver is a computer connected to the Internet which hosts one or more websites. But a server can do much more. While webserver software is listening to port 80 for connections, many other services can listen to other ports. A computer connected to the Internet can be webserver, *FTP* server and mail server at the same time: the different services each listen to a certain port. Webserver software listens to port 80, FTP server listens to port 21. (FTP = File Transfer Protocol)

We already used our browser to connect to port 80, but we could try to find out more about our target server by trying other services a server may have running.

3.3.1 Telnet

Using *Telnet* could certainly learn us something about the target server. Launch 'telnet' and try to connect to the target server. With the Telnet program, you can connect to any port you would like, but we choose to connect to the standard telnet port - 23. You'll now see something like this:

```
Digital UNIX (target.com) (ttyq6)

login:
```

You can now try to enter a dummy name and a dummy password, but the chance that your name and password combination will be correct isn't very large, no need to explain that...

But we certainly have learned something about our target: our target server is running UNIX, Digital UNIX if you want to know exactly. Trying this could also reveal that you're dealing with a Windows NT server, which makes a great difference. It's also possible that you aren't able to connect like this and that the connection is immediately closed with an error message. In that case, Telnet won't help you, but it was worth trying.

Please note There's an important difference between the Telnet program, the Telnet protocol and the Telnet port. The Telnet program is just telnet.exe which comes with Windows (or any other operating system), the Telnet protocol is a set of rules about communication between two computers, the Telnet port is (in most cases) port 23 which is used for Telnet (the protocol!) connections.

3.3.2 SSH

If Telnet refuses all connections, it might be possible that users are only allowed to use *SSH*. SSH stands for *Secure Shell* and does the same as Telnet, but all data is encrypted before transmission and decrypted afterwards. Besides, authorization methods available with SSH are much safer than those of Telnet.

To find out if a server supports SSH, just telnet to port 22. If you are able to connect to this port and you see something like `SSH-1.5-1.2.26`, this means that SSH is enabled. It's good to know that SSH is enabled, but it won't help you much, because SSH it too well protected for us. We won't be able to use it for our purposes. When you now press enter, the remote host will close the connection because you didn't send data the SSH protocol expects. It will say: `Protocol mismatch.` and close the connection.

3.3.3 FTP

You probably already know something about *FTP*, but I'll explain anyway. FTP is just another protocol - like HTTP, Telnet and SSH. FTP stands for *File Transfer Protocol* and is used to transfer files. FTP server software is listening to port 21 in a common server.

We can use FTP for the same thing as Telnet - we can try to gain more information about our target. What we first do is launch the Windows 'telnet' application and try to open a connection to target.com port 21. If telnet is unable to establish a connection, target.com is not a FTP server.

On the other hand, if telnet actually is able to connect to port 21, FTP server software is running. We could try now try to login as a normal user, but that's just as useless as trying to login using the Telnet protocol when we connect to port 23: we don't know the right username - password combination and we would have to be very lucky to guess correctly.

But, many FTP servers also allow people to login anonymously and that's exactly what we're going to do. Type `USER anonymous`, press return and see what happens... If our target server supports anonymous login, you will now see something like 'Guest login ok, supply your full email address as password'. Well, that's obviously what we're not going to do. Type something like `PASS noone@nowhere.com` and press return again. At this point, you should see some login messages saying you logged in successfully.

Alright, now that we know that logging in anonymously is allowed, we close down our Telnet application and launch our favourite FTP client. I prefer WS_FTP, but any FTP client could be used. Use this FTP program to connect to target.com, make sure that the FTP client logs in anonymously without using your real email address as password!

Once logged in, you should have a good look at the directory structure at the target server and see if you can find any files which could help you gaining access to the website hosted at the very same server. (A file containing all usernames and passwords would be very useful, for example.) You should also try to find out how well the server is protected: which directories do you have access to and which directories can't you access? And can you reach the directories where the website is hosted? (If yes, it might be possible to download the whole site - including protected area. But this would be pretty stupid of the system administrators...)

4 Server-side protections

Since our little exploration with our browser, we know what type of protection we're dealing with. It's now getting type to explain a little more about each of these protections and how to deal with them.

4.1 Webserver software authorization

Let's start with the best protection. After that, everything else will be easy, don't you think? As I already said, when a box pops up asking you to enter your username and password and an 'Authorization failed' page appears when you click 'Cancel', you know you're dealing with such a protection. Let's examine how a system administrator achieves this.

It depends on the webserver software and its configuration if it's possible to use this technique to secure your website. Therefore, I don't promise you that it'll work on your webserver. On the contrary: this technique doesn't work on most normal ISP and free webhosting accounts. If you want to try it yourself, you should think about downloading and installing the free webserver software

Apache (see section 8). It's the most common webserver software and supports the technique I'll describe.

4.1.1 The .htaccess file

In the Apache webserver configuration files, it's possible to define a filename for authorization settings. After doing this, each time a website visitor requests a certain URL, the first thing the webserver does is scan the directory of that URL (and all higher directories) if there is such an authorization settings file.

The default name for this file is *.htaccess*. It may seem strange that this filename starts with a dot, but that's because Apache has originally been developed as UNIX software. Under Windows, it's possible to give files and directories a special 'hidden' flag. Under UNIX, starting a filename or directoryname with a dot does exactly the same.

When an authorization settings file is found, the webserver will process the directives in that file before returning the requested webpage. Please note that I wrote 'and all higher directories': subdirectories of a directory with authorization settings automatically have the same access rules.

Alright, let me give an example of this file (which is a normal ASCII file).

```
deny from all
AuthType Basic
AuthName "*** Secured Membersection ***"
AuthUserFile /home/username/users.pass
require valid-user
satisfy any
```

It's possible to set this up a little different, but this is the most common way. First, you deny access for everyone. After that, you define the authorization type, a text users will see when the 'Authorization required' box pops up and the full path to the file containing the valid user-password combinations. Finally, you tell the webserver to satisfy any valid users: return the requested file(s) if the password is valid.

Since authorization settings are always in the secured directory itself, it's impossible to read this using your browser without having access to the secured webpage as well. On the other hand, as you can see, the userfile can be in any directory. Therefore, the userfile could be in an unsecured directory, although this would be rather stupid.

We now know more than enough about this file, the next file we'd like to examine is the userfile (which can have any filename). On UNIX systems, a userfile looks like this:

```
Steinowitz:XDOdblTuGoGJA
DrEaD:CFI65HBCHifxE
```

As you've already noticed, each line is in this format:
username:encryptedpassword. The encryption used here is done by the standard UNIX *crypt* function. Both usernames and passwords are case-sensitive. Since this function isn't available on Windows systems, some Windows webservers don't encrypt the passwords. If we would only have the userfile…

4.1.2 Intermezzo: the UNIX *crypt* function

The UNIX *crypt* function is used by a lot of UNIX software, especially software which provides Internet services. FTP server software also has userfiles and which function is used to encrypt the passwords? The standard *crypt* function! ISP dialup servers also have large userfiles and which function is used to encrypt the passwords? You already get it! Of course, there are many exceptions, but there are many more cases for which this is true. All those files are in the same format I described above. (There may be more different fields in each line to store more detailed account information, but the first two are always username and encrypted password.)

If you have access to a UNIX server using FTP, you should try to get the file */etc/passwd*. Believe me, even anonymous login may give you access to this file. Find one, open it and you'll immediately recognize the encryption used.

Since this function is used that much, I don't even have to mention the fact that there've been many people who studied it. The funny thing about it is that you can't retrieve the encrypted text once you've encrypted it.

For example, when webserver software needs to check if a password is valid, it encrypts the password given by the user en compares that to the already encrypted password in the userfile. No match, no valid password. The webserver software simply cannot decrypt the password in the userfile.

(If you forget your password, the only way to gain access again is to set a new password using the crypt function. When you have Telnet access to a UNIX server, an easy way to add/update users to a userfile is to use the *htpasswd* program, if it's installed.)

The *crypt* function itself is very easy to use: it only requires two parameters. The first one is the text to encrypt. The second one is called the hash and must be two characters. These characters are used to encrypt the first parameter. It's easy to get to know which hash was used, since the first two characters of the encrypted password are the hash.

If you want to know exactly how the function works, you should get yourself an open source UNIX/Linux distribution. The function is always available in these distributions and since it's open source, you can have a good look at the source code.

4.1.3 Gaining access

We now know a lot about how this protection works, but we still don't have access to the secured webpage. I'll mention three possible ways to gain access, althought I won't mention the details. They're general approaches, not detailed step-by-step tutorials. If you want to know more, you'll have to search the Internet for more information, since I won't tell you. All I'll do is give you a couple of URLs in the resources section (section 8).

- Find a program which bruteforces all (or many) possible username-password combinations. All you need is such a program, the target URL, a (fast) Internet connection and (a lot of) time. Note that the system administrator might notice that you've been requesting the same secured page many times. Therefore, you should use a secure proxy server. (See section 6.3.)

- Use (anonymous) FTP to gain access to the webserver and see if you can get the userfile (or other interesting files, the secured section itself for example). Use a program which bruteforces these userfiles. Also takes a lot of time, but you don't need to be online all the time because you have the userfile on your own computer.
- Find another way to gain access to the server and retrieve the userfile or modify/remove the authorization settings file to gain access. If you know more about hacking, you might be able to use Telnet for this.

None of these solutions is very easy to accomplish. If you haven't much experience with these kind of things, you might want to find other websites using other protections first.

4.2 CGI protections

The last subsection covered a very specific way to secure a website. It's a very straight method: you either do it like the manual says or you don't do it at all. This subsection, on the contrary, covers a very broad range of ways to secure your website. There is no convention or rule which says: when using CGI to protect your website, you should use the function `secureWebsite`, simply because there is no such function. If you want to use CGI for securing your website, you'll have to provide all necessary features yourself.

It may seem that this is more difficult to handle than webserver software authorization, because every CGI protection is different. But that isn't true, simply because webserver software is in full control - CGI programs aren't. As a matter of fact, CGI programs are controlled by the webserver software.

When a browser submits a form to a CGI program, a new instance of the CGI program is started by the webserver software and this CGI program is able to 'read' the data supplied by the browser. (And the browser got it from the user, who provided the data in input fields like textboxes and checkboxes.)

After reading this data, the CGI program (which runs on the server, I'll mention this once again because it's very important) processes the data and does whatever the coder wanted it to do. The CGI program could read and write files and databases on the webserver, it could modify webpages (HTML-files) and so on. Before terminating, the CGI program is obliged to return something.

HTML code The CGI program could return a webpage like any other. For example: search engines (which are CGI programs) return webpages containing X hits. When you click the 'next' button, a new instance of the CGI program is started and the program returns the next X hits.

Picture A CGI program could also return a picture. In most cases, this is a GIF file, because it's easy to manipulate GIF images with CGI programs, but it could also be another filetype. Example: graphical counters.

URL Redirecting a browser using CGI programs is easy, provide a new URL (which can be a webpage, a file or any other URL) and the browser will go there.

Plain text Hardly ever used, but it's possible. One would rather return HTML instead of plain text.

4.2.1 CGI programs

A CGI program can be written in almost any language, as long as it's possible to run it on the webserver. Therefore, it depends on the operating system of the webserver in which programming language CGI programs are coded. Languages often used for CGI purposes and supported by both UNIX and Windows NT servers are C and Perl.

If you'd like to do some CGI coding yourself and you haven't got much experience with C or C++, you should learn Perl, because that's much easier to learn.

C programs can be written on any computer, but the source code should then be uploaded to the webserver and compiled using Telnet if you want it to be a CGI program. Perl programs, however, don't require to be compiled, because Perl is a scripting language and Perl interprets the scripts when started.

CGI programs end with the extension *.cgi* most of the times, but this isn't required by all webservers. It depends on the configuration. Most large search engines and similar services don't use this extension.

A well-configured webserver doesn't give website visitors the possibility to actually see a CGI program: whenever a CGI program is requested, the webserver executes the program and sends the output of the program to the website visitor. Therefore, you don't see the source code of the Perl script, neither do you see the compiled C program.

A stupid CGI programmer who codes in C, however, could upload the source code to the webserver, compile the program... and leave the source code where it is! So if you know the name of a CGI program and you think it could be written in C, you could try the following:

CGI program: *http://www.target.com/lamer/myprogram.cgi*
Source code: *http://www.target.com/lamer/myprogram.c* ???

You're rather lucky when *myprogram.cgi* is a security program and you get the source using this trick. Or, to say it a little different: the webmaster is very stupid!

4.2.2 Possible techniques

What I'll do now is mention some techniques a CGI coder could use to prevent unauthorized website visitors from accessing his secured section. You'll see that it can be fairly easy to evade such protections sometimes.

Redirection One way is to check the username and password by looking them up in a database (which can be a normal textfile) and then redirecting the user depending on the result of that. If the password is incorrect, the user is redirected to a 'wrong password' page and if the password is correct, the user is redirected to the secured webpage (a normal HTML file). If this is the only technique used, it's easy to access the secured webpage: all you have to know is the URL of the secured webpage. More on this in section 6.5.

More CGI programs Another way is to make CGI programs of all secured pages. If you do it this way, you can ask for username and password once and pass them on to each new secured page subsequently visited (which are all CGI programs). The password can now be checked by each CGI program before returning the contents of the secured page.

One CGI program It may sound a little strange, but it's possible to combine a whole section of a website with one CGI script. This makes it easier to validate username and password, because you only need to pass them on to the same program (contrary to the previous possibility I mentioned). On the other hand, it's very annoying if you have to put many different pages into one CGI program: it becomes harder to oversee the source code. Therefore, one would rather put the contents of the secured pages in different datafiles which the CGI programs will read and return to the user. And that will make it easier for us to access them: all we need to know are the names of those datafiles.

Cookies It's also possible to set a cookie when a username-password combination is valid. All secured pages should then request the cookie and check the contents (which should be a certain code) to see if the user is authorized. Checking these cookies could be done with JavaScript, but this isn't a very safe way, which we'll see later. CGI programs are needed again to do it more secure.

I think these are the most common techniques used to secure a website using CGI programs and I must say that the webserver software technique is much better. One positive point of CGI techniques is that it's possible to use the UNIX *crypt* function with the most common CGI programming languages.

5 Client-side protections

I already said that client-side aren't very good protections, so I won't spend too much time on this section.

5.1 Java applets

Most Java applet protections are useless, because they don't provide any security. There are only a few things you should know about Java...

First of all, Java source code is first compiled (by its author) to *byte code*. This results in *.class*-files consisting of byte code. These files can be implemented in websites with the <APPLET>-tag I described earlier. When your browser loads the applet, the byte code is interpreted by the Java Virtual Machine installed on your computer (which depends on the platform!). After that, the result of this (= machine code) is executed by the processor.

Very interesting to know is that byte code can be decompiled. So all we need are 1) the *.class*-files of the applet used to protect a website, 2) a Java decompiler and 3) enough Java knowledge to understand the Java source code of the applet.

- The *.class*-files used by the applet can be easily found: after downloading them, your browser automatically saves them in its 'cache' or 'temporary internet files' directory. Look for them in those directories and copy them to another directory.

- A Java decompiler. Well, search for it on the web. A very good one is called JAD. You should be able to find it.
- Java knowledge. Sorry, that's one thing you'll have to do yourself. But even if you never coded in Java, just have a look at the source code: in some occasions, Java knowledge is hardly required.

Use your Java decompiler to decompile the *.class*-files to files containing the Java source code, read them carefully and you'll know what to do.

There is one exception on this: it's possible that the Java applet sends data back to the server and that the server subsequently sends data to the applet again. If you're dealing with such a protection, it's not fully client-side anymore. Part of the protection (probably the most important part) is located on the server and it depends on the source code what you should do in this case.

5.2 JavaScript protections

In the previous subsection, you needed more than just knowledge. In this subsection, all you need is JavaScript knowledge. Open the HTML-file your protection is in, look at the JavaScript and try to find out how you can gain access. Learn how to reverse!

It's possible that (part of) the JavaScript code is locate in a separate file with the extension *js*. When that's the case, you'll notice this when reading through the HTML source code. I don't need to explain that you need this file if you want to know how the protection works.

Sometimes, a JavaScript protection works like this: you provide a username and password. With this username and password, some JavaScript functions calculate the filename of the secured website you want to visit and change the location of your browser to a new URL using this filename. In this case, you can't see what the correct username - password combination is: when you enter a wrong combination, you'll be redirected to a non-existing URL and you'll get the famous 404 error message. You'll have to do some bruteforcing here, see 'Bruteforcing remote filenames' (section 6.5).

6 Tips and tricks

In this section, I'll mention some tips you could use to your benefit. It would take me too much time to explain every little detail of all these things. If you think it would be very useful or if you want to learn more about it for other reasons, search the Web for more info on it. Believe me, it's easy to find a lot of information on each of these subjects.

6.1 Using CGI

I already explained what CGI is and what webdevelopers do with it, but there's more. In certain cases, you could abuse it in very nasty ways. Generally, there are two ways to do this.

- The first way to do this is without writing your own scripts. When programmers write bad CGI programs, you can abuse this by providing other input for the scripts then expected. In many cases, input from the webuser isn't validated very well. And under normal circumstances, this doesn't really matter. But this gives us the opportunity to make CGI scripts execute more commands then normal (in some cases, we can even view the password file). This could really help us with gaining access to a secured website. And there are some other possibilities...

- The second and last way to do this is to learn writing CGI programs yourself. When you know CGI, you could write programs posting the same data all over and over again and you could do other things automatically which would otherwise take a lot of time.

Search for CGI, Perl, C, exploits and such terms. If you want to do such things, I highly recommend you to learn how to code CGI programs yourself.

6.2 Using email

Some mailboxes are used for tasks which have to be done very often and in most cases, programmers write certain 'bots' which automatically answer all incoming mail for these mailboxes.

One example are the `forgot_pw` mailboxes (don't remember the name exactly) used by free email services like Hotmail and Yahoo. When you forget your password, you got to the website of the email service, you submit a certain form, the data is passed on to a CGI program and this sends a message to the forgot_pw mailbox. The automatic reply contains the password of the right account. When you know in which format this CGI program sends the message to the forgot_pw address, you can simulate this and get any passwords you want! Of course, this is only an example and since it was posted on several messageboards, I think email services will all have corrected this now.

Use your imagination if you want to benefit from this approach.

6.3 Using a secure proxy server

When you use a (secure) proxy server, you don't connect to the remote host immediately, but you connect to the proxy server. After that, the proxy server connects to the remote host and returns to you what you wanted.

The reason that these proxy servers exist is to improve the speed at which you can retrieve pages often visited pages. Proxy servers have a cache containing often requested files, which makes it unnecessary to connect to the remote host for all files asked for: files still in the cache can be returned to the user immediately.

But there's a far more important reason for using a proxy server: it strongly increases your anonymity while surfing! There's been written quite a lot on anonymity on the Internet and it all comes down to the fact that you are not anonymous unless you use a good proxy server.

How this works? Well, you don't connect to the remote hosts anymore, but the proxy server does! So your hostname and IP address can't be logged at the remote host, only the hostname and IP address of the proxy server.

See the 'Additional resources' section (8) for a nice 'public proxy' site you should visit.

6.4 Portscanners

A portscanner is a program which scans ports, but that won't surprise you. There are many portscanners and they all work a little different, but the most important about it is that you can supply a hostname or IP address (or a range of IP addresses) and a range of ports. After doing that, the portscanner will try to connect to each of these ports and let you know if it could connect or that it couldn't.

Doing this, it's easy to see which IP addresses are running a FTP server (connected to port 21!) or a webserver (connected to port 80!). But there's more. I already said that these ports are just a standard, it's easy to configure webserver software in such a way that it listens to port 300. Well, we'll notice when we do a portscan from port 1 to 500 (for example). When the portscanner tries to connect to port 300, it connects and when you check the text the server returns, you will notice it contains 'HTTP'. And wasn't that the protocol used by WWW-servers?? Exactly! We immediately launch our browser (Netscape, in my case) and visit the url *http://www.target.com:300/* It could be that we've found a hidden website...

Search for sites on hacking and you'll find many portscanners. I only gave an example of what you could use it for. There are many other possibilities: learn more about networking and use your imagination.

6.5 Bruteforcing remote filenames

One of the easiest ways to hide a part of a website for the normal visitor is to give the files certain names and don't let people know what these names are. If you have a normal website at *http://www.target.com*, it's easy to make a subdirectory and put another, hidden website there: *http://www.target.com/hiddenwebsite/* If you don't link to it, your visitors won't know about it unless you tell them!

And, naturally, the same counts for filenames: noone would try *http://www.target.com/anotherwebsite.html* if you wouldn't link to it and you wouldn't tell anyone.

But, there is a way to get to know where these hidden files are. The most obvious way is to hack yourself into the server, go to the webserver directory and have a good look at the filenames there. The other way is simply to bruteforce it: try all possibilities. You understand that you shouldn't do this yourself, because it'd take way too much time. Write a program which does it or search the web for a program which does it for you.

Note: You wouldn't do this when you don't know *if* there is a hidden website like I described, because even when a program does the job for you, it takes quite some time.

7 Last words

I hope you enjoyed reading this essay as much as I enjoyed writing it. Furthermore, I hope you learned something. And don't forget: I'm not writing this for money, but I do ask something for it in return. I ask you to do the same as I do: share your knowledge with everyone who wants to know!

Steinowitz
switz@newmail.net
http://dread99.cjb.net

8 Additional resources

- Apache (webserver software)
  http://www.apache.org
- Public proxy servers list
  http://freebooks.hypermart.net/proxy/proxies.htm
- Anonymizer (surfing the web anonymous)
  http://www.anonymizer.com
- AltaVista (search engine)
  http://www.altavista.net
- Perl
  http://www.perl.org
- EliteSys (secured website bruteforcer)
  http://web.idirect.com/ elitesys/index.html
- HackerLair
  http://www.hackerzlair.org
- HackersWeb
  http://www.hackersweb.com
- HackPalace
  http://www.hackpalace.com/usa/
- L0pht
  http://www.l0pht.com
- RootShell
  http://rootshell.com
- United Hackers Association
  http://www.uha1.com

Disclaimer notice

I wrote this document for **educational** purposes only. I am **not** responsible for anyone abusing the knowledge I shared with others by writing this.