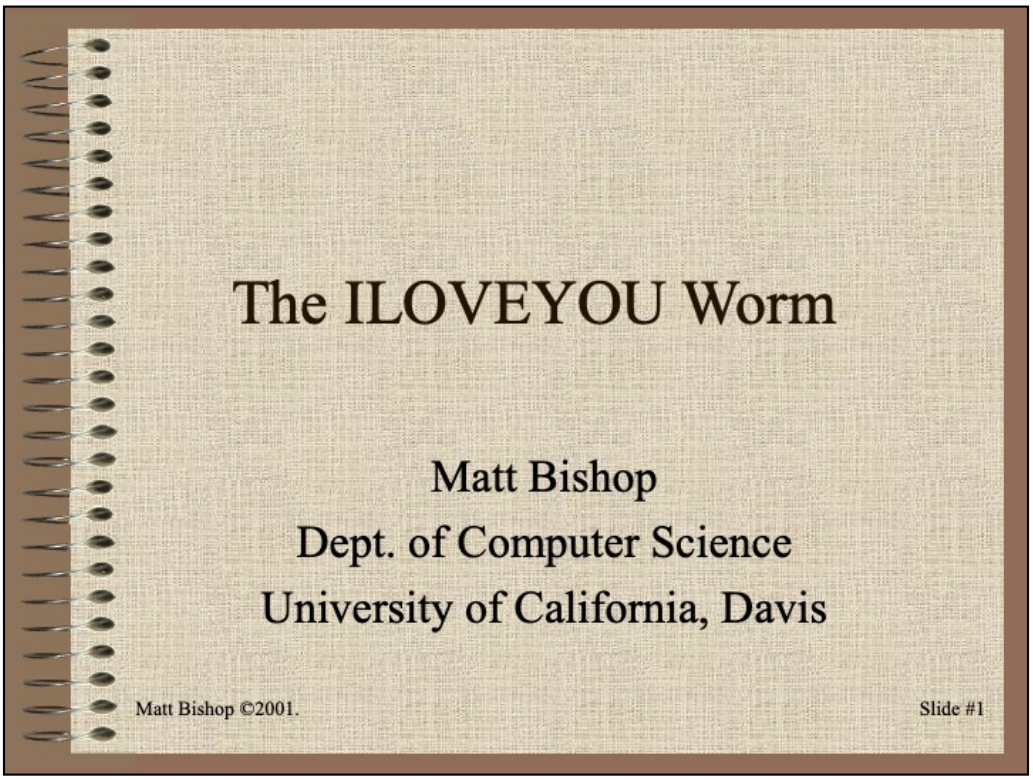
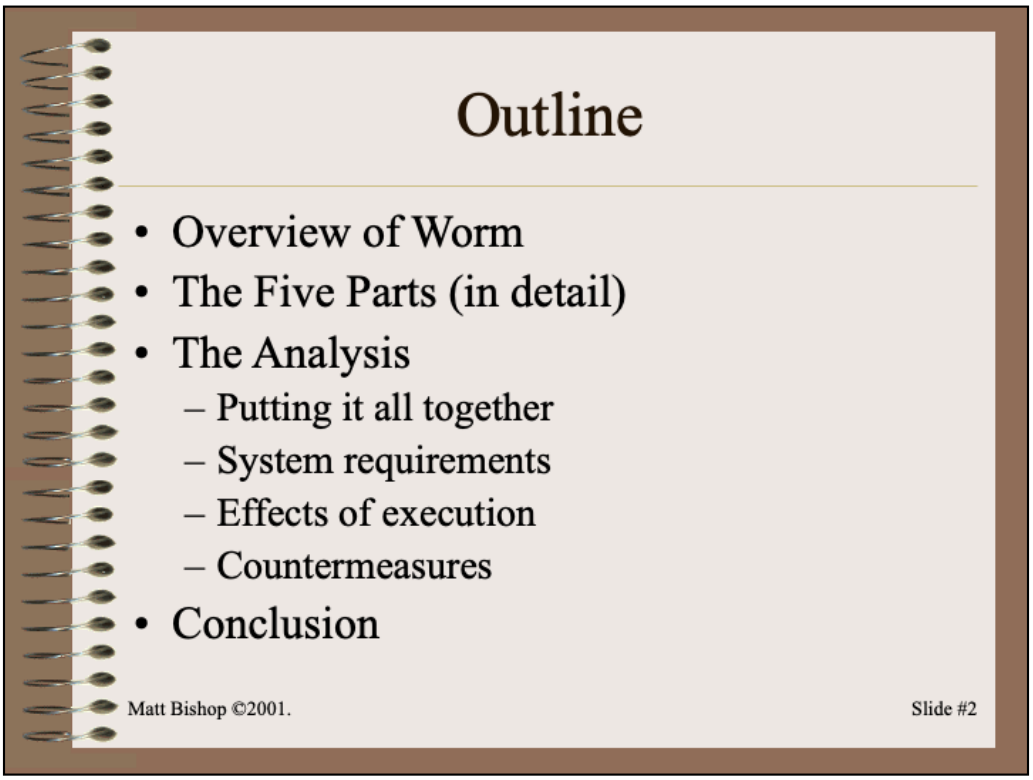



The ILOVEYOU worm struck on May 4, 2000. It came as an attachment to email. The letter asked readers to click on an attachment to read a love letter. The attachment contained a Visual Basic program that Microsoft Outlook interpreted as commands, and executed.

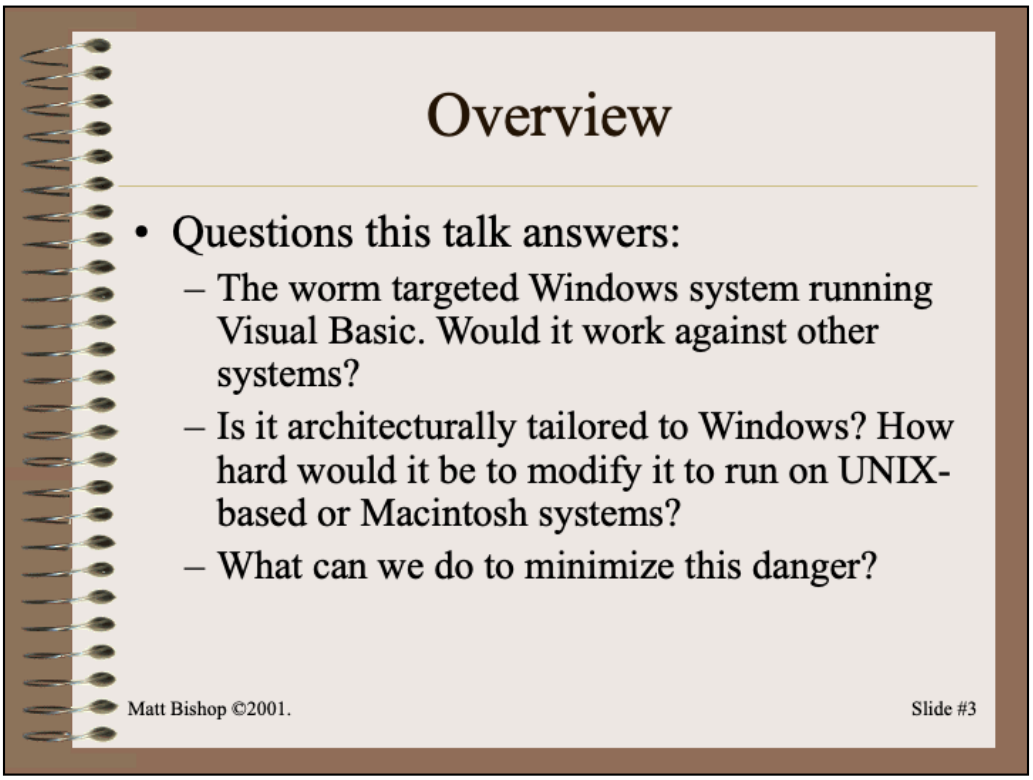
The worm spread throughout the world very quickly, affecting the British Parliament, the U. S. Congress, the U. S. Air Force, and innumerable businesses and organizations. Filters to block the mail were quickly developed and installed, but a spate of copycat worms in the next few days (for example, one entitled “JOKE” and another with an invoice that would be billed to the recipient’s credit card) evaded the filters.




The worm was composed of four parts: the *main* routine, the *regruns* component, the *html* component, the *spreadtoemail* component, and the *listadriv* component.

After looking at these, we'll put it all together to describe how the worm did what it did. This will also give us the requirements for a sstem to execute the worm, as well as the effects of executing it and the countermeasures you could take against it.

We'll conclude with some comments about the worm on non-Windows systems.

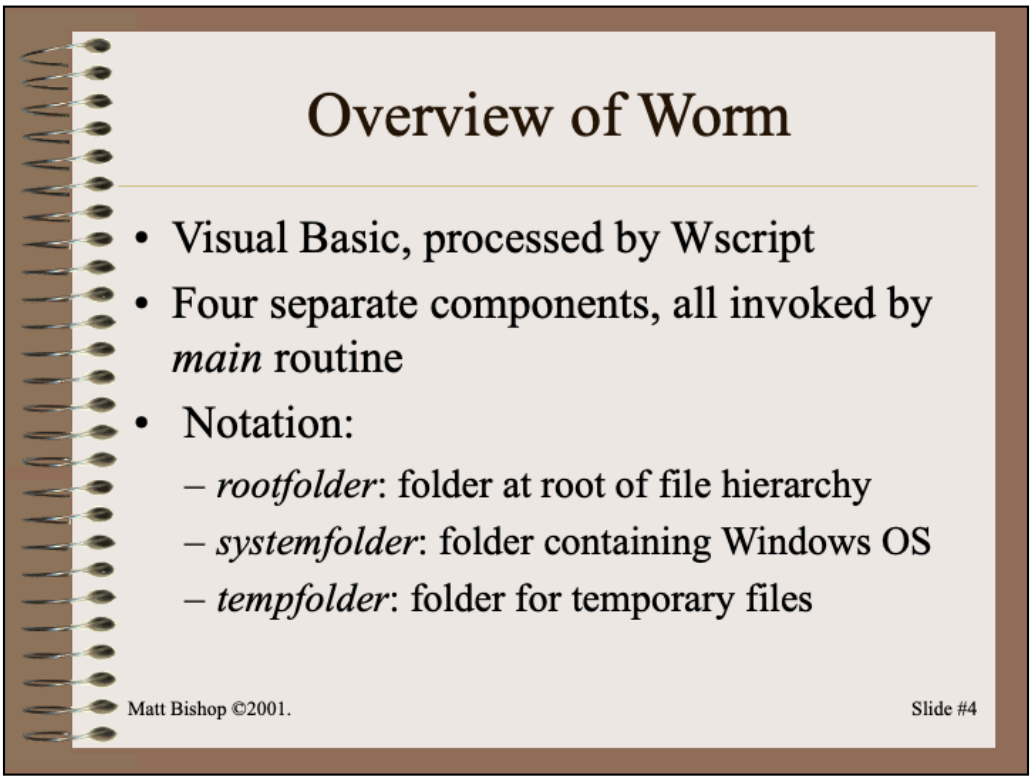



This talk looks at the worm, but as an object lesson in malicious logic (code which does not comply with a site's security policy). This leads to several questions:

The worm invoked *wsh*, a Visual Basic interpreter for Windows, and updated the Registry explicitly. Other systems, such as the Macintosh, handle state information differently (basically, there is no Registry; the Preferences folder in the System Folder serves a similar purpose). Would the worm work against other systems?

How hard would it be to modify the worm to work against UNIX-based systems? Could something similar work (there is a widespread belief that UNIX-based systems are immune to worms and viruses because of their multi-user protections)?

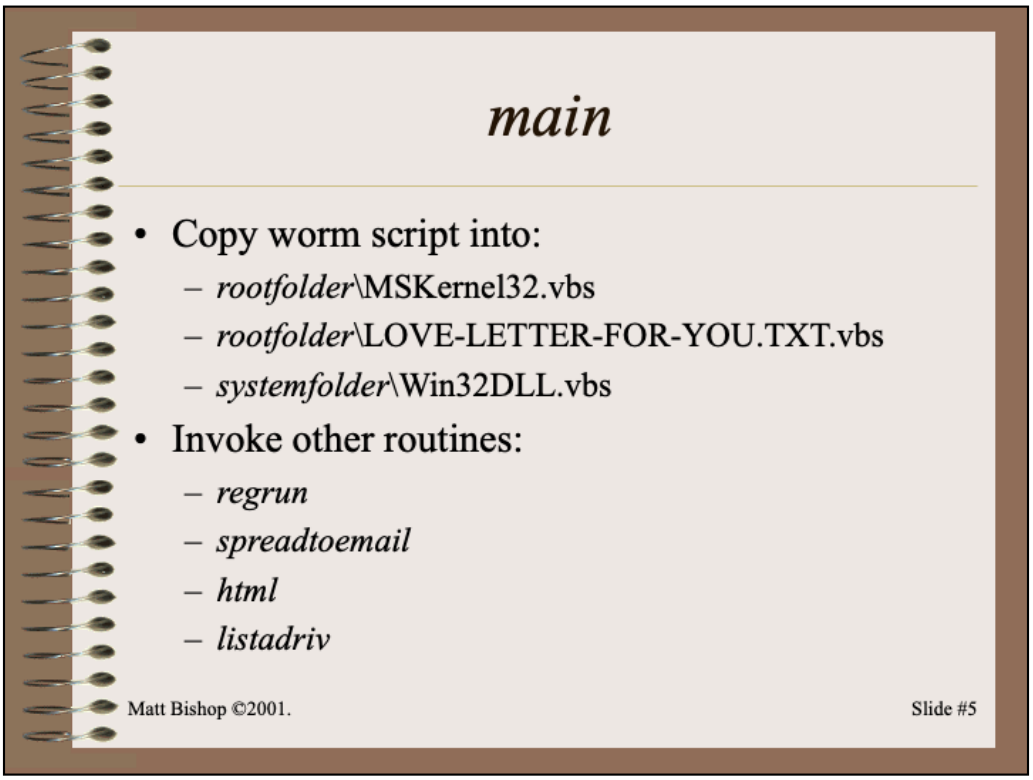
How can we (or *can* we) protect ourselves against things like the worm?




The worm uses the Wscript interpreter to run the Visual Basic. That's why you have to click on the attachment; it needs to be handed off to the interpreter.

The *main* routine does some limited setup (including modifying the Registry).

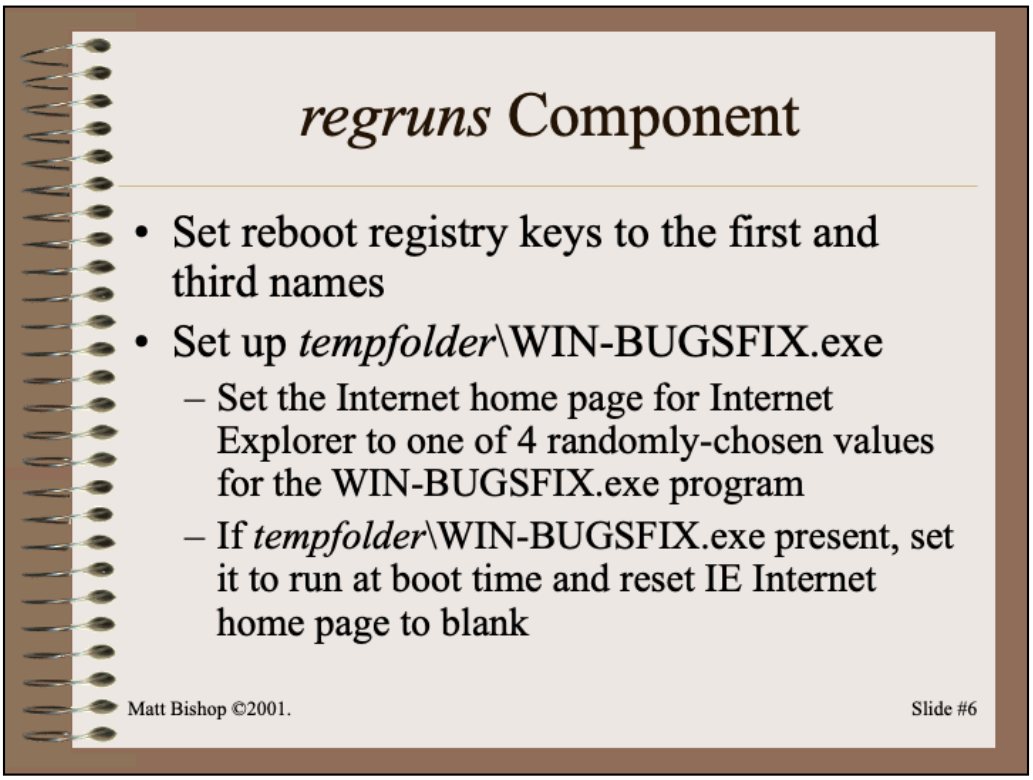
The values for *rootfolder*, *systemfolder*, and *tempfolder* are obtained from the system (using *GetSpecialFolder*), so changing these from the standard locations does not help.




The *main* routine copies the worm into three places for later use. The first two will be invoked when the system reboots (see *regruns*) and the third in *spreadtoemail*.

Note the names. The first one looks like a new kernel program (MSKernel = MicroSoft kernel) and the third a DLL (unless you notice the DLL is not an extension, or is visible). The second also looks like a text file (.TXT) because Windows would hide the extension. If you viewed it in another way, the second extension would be visible (and a give-away that something is seriously wrong).

The other routines are invoked in the order shown. The next slides summarize what they do.




The Registry keys are set so when the system reboots, the worm is restarted. The keys are:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\MSKernel32

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices\Win32DLL

and they are set to the first and third names, respectively.

The next step requires the \systemfolder\WinFAT32.exe file, and is skipped if that does not exist. The registry key is:

HKCU\Software\Microsoft\Internet\Internet Explorer\Main\Start Page

and is set to any of the following four values:

•http://www.skyinet.net/~young1s/HJKhjnwerhjkxcvytwertnMTFwetrdsfmhPnjw6-587345gvsdf7679njbvYT/WIN-BUGSFIX.exe

•http://www.skyinet.net/~angelcat/skladjflfdjghKJnwetryDGFikjUIyqwerWe54678-6324hjk4jnHHGbvbmKLJKjhkqj4w/WIN-BUGSFIX.exe

•http://www.skyinet.net/~koichi/jf6TRjkcbGRpGqaq198vbFV5hfFEkbopBdQZnm-POhfgER67b3Vbvg/WIN-BUGSFIX.exe

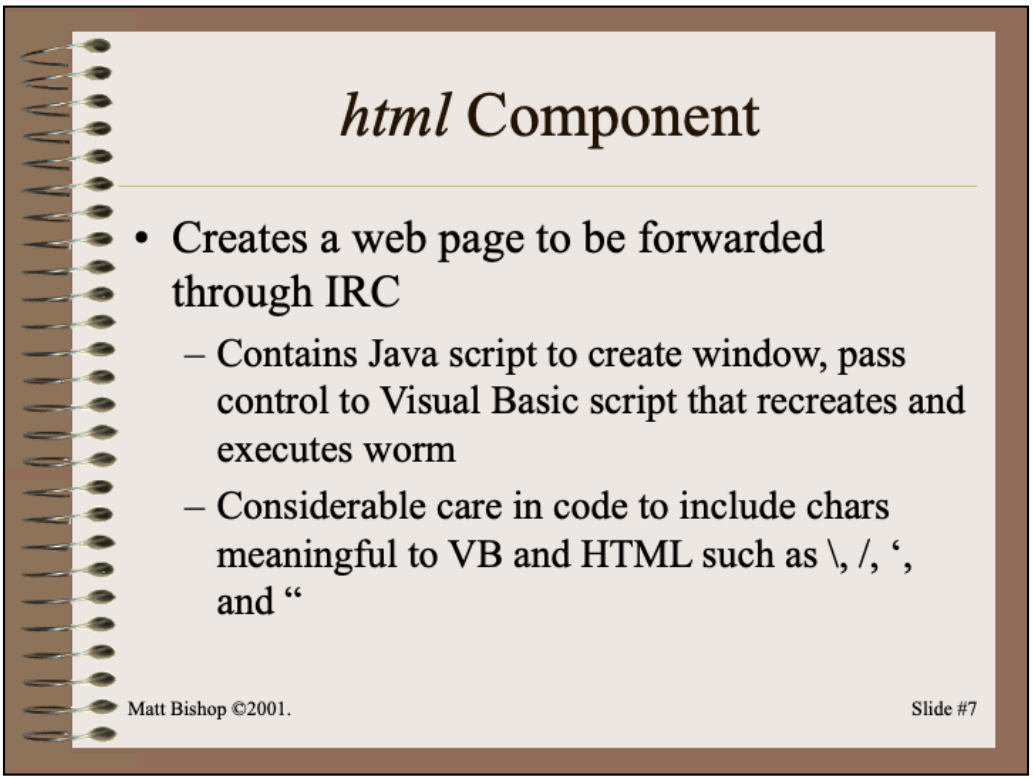
•http://www.skyinet.net/~chu/sdgfhjksdfjklNBmnfgkKLHjkqwtuHJBhAFSDGjkhYUgqwerasdjhPhjasfdglkNBhbqwebmznxcbvnmadshfgqw237461234iuy7t

hjg/WIN-BUGSFIX.exe

The Registry key for WIN-BUGSFIX is:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\WIN-BUGSFIX

and is set to the WIN-BUGSFIX.exe program.



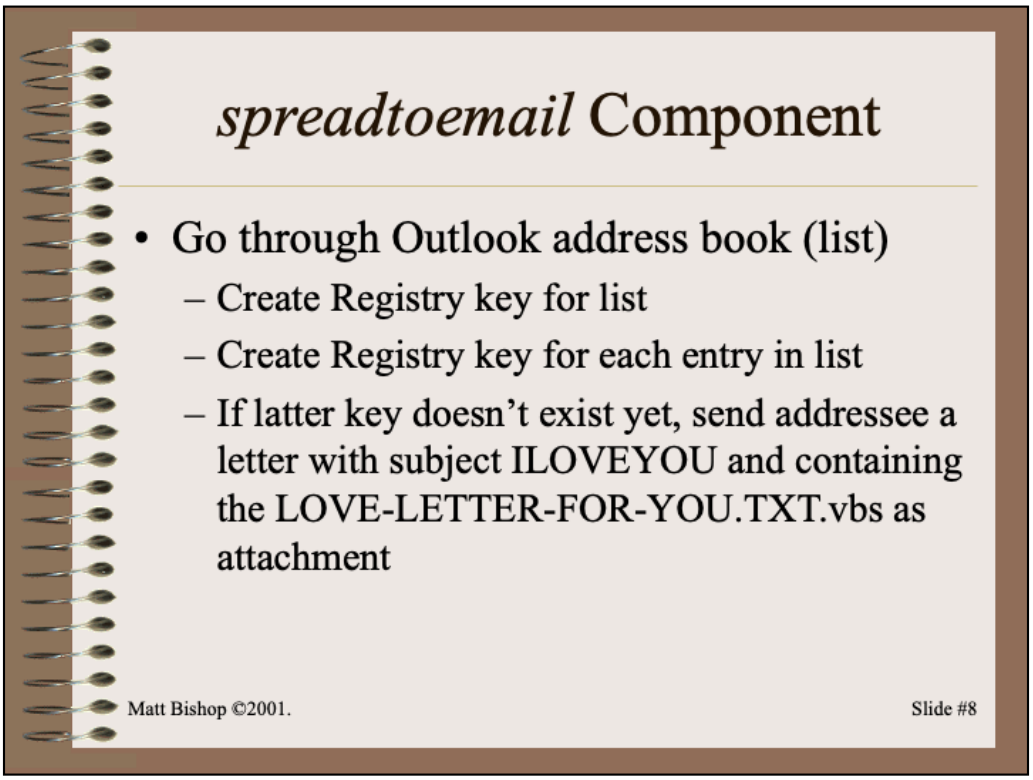

The web page is systemfolder\LOVE-LETTER-FOR-YOU.HTM, and the next routine forwards it through IRC. It places the worm code into two (long) strings. First, it changes the following sequences of characters:

?-? to /  #-# to ‘  @-@ to “  ^-^ to \

It then splits the two strings into many shorter lines, maps

‘ to [-[  “ to ]-]  \ to %-%

adds quotes and CRLF to each line, and writes them into the web page file.



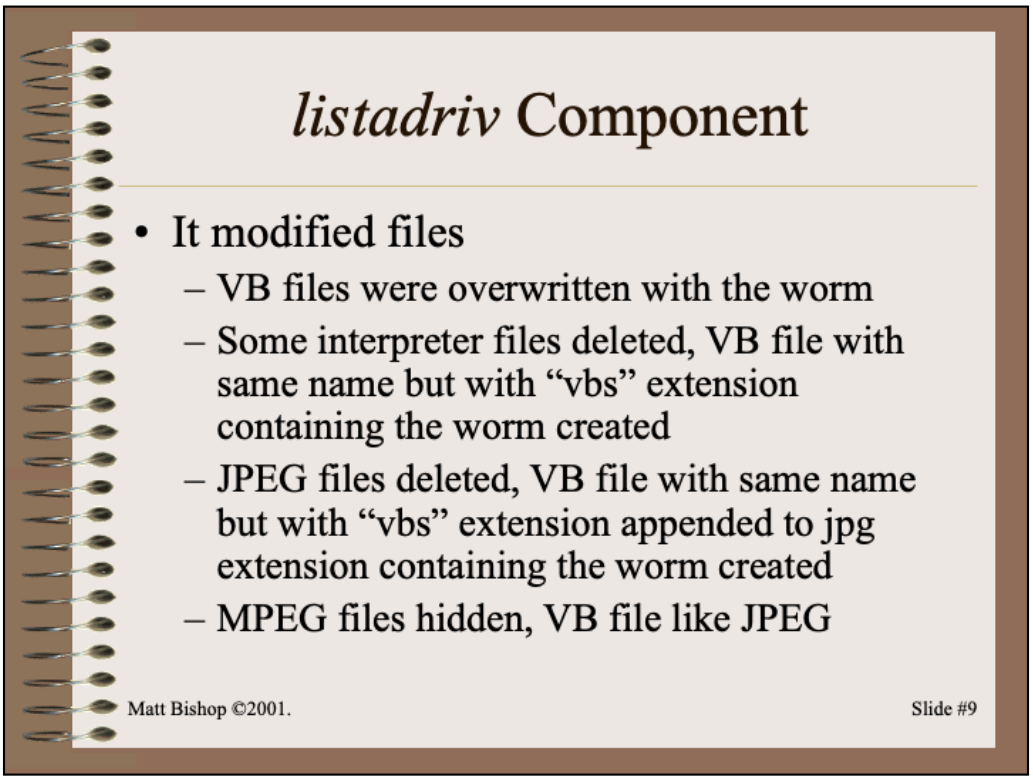

The Registry key for the list is:

HKEY_CURRENT_USER\Software\Microsoft\WAB*listname*

and for each address in the list is:

HKEY_CURRENT_USER\Software\Microsoft\WAB*addressee*

This routine is the primary vector for spreading the worm. It also tries not to send the letter to the same place twice; however, as most people were in multiple user's lists, they got many copies anyway.




This recursively descends both local and remote drives.

The exact mappings for modification:

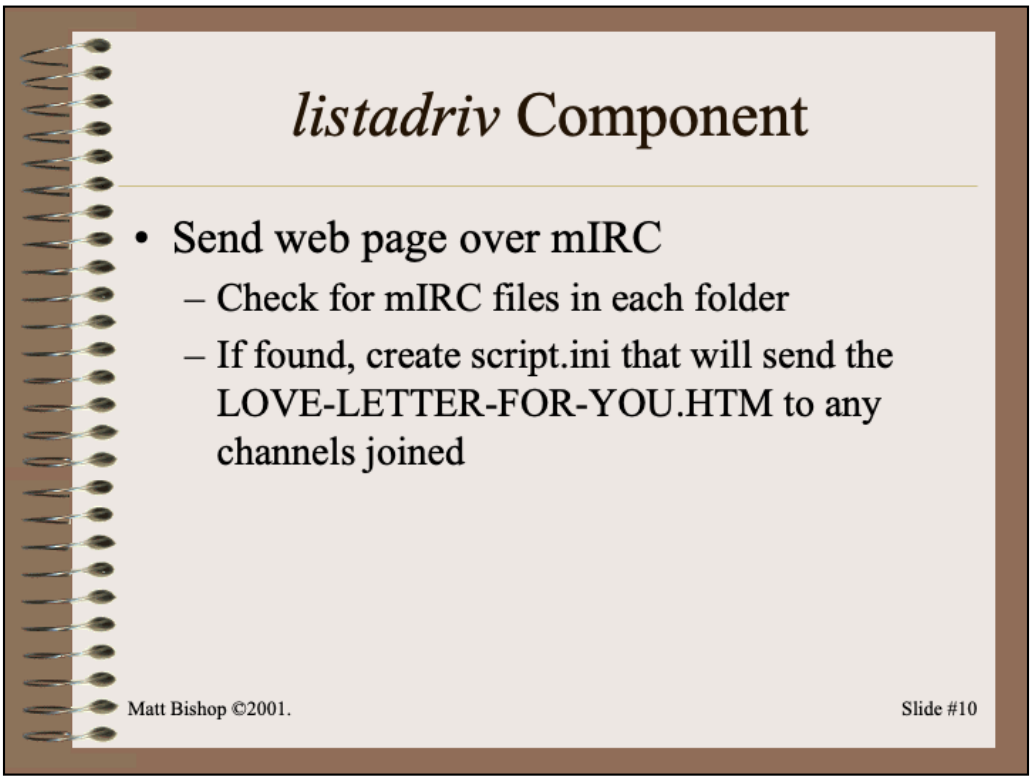
vbs, vbe are the Visual Basic extensions

js, jse, css, wsh, sct, hta are the extensions to the interpreter files

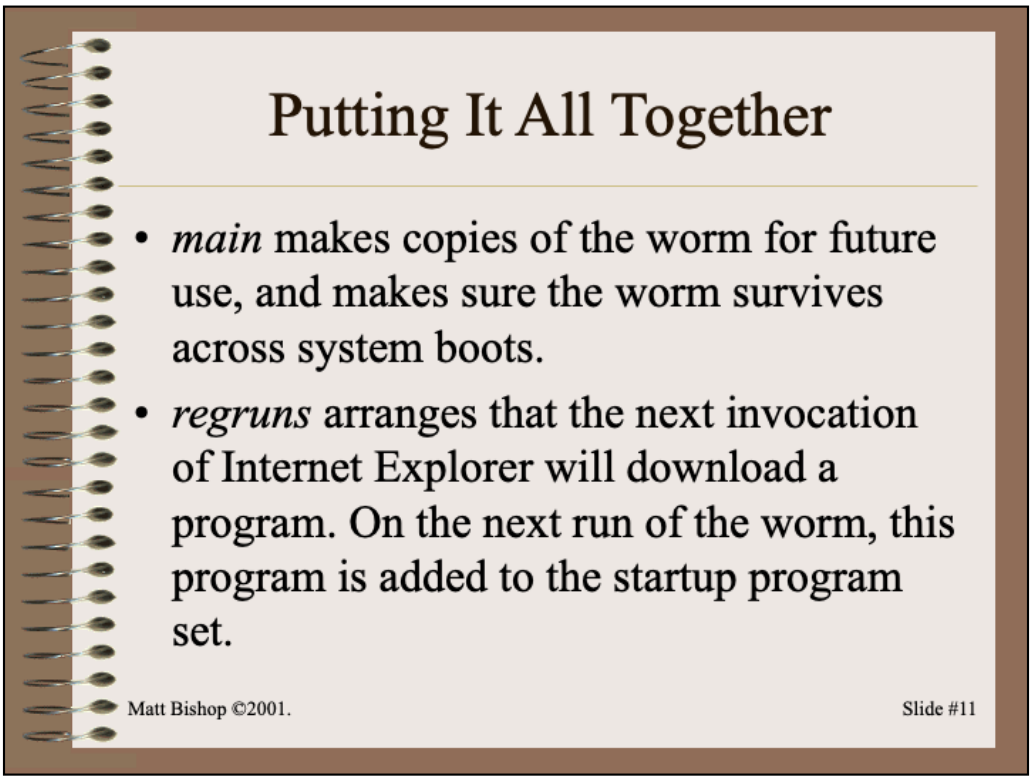
jpg, jpeg are the JPEG extensions

mp2, mp3 are the MPEG extensions

Unlike the other three types, the MPEG extensions have 2 added to the file type. As an MPEG file should be a normal file, this turns it into a hidden file. But if the attribute is something else, it won't be hidden (who knows what it will be?)



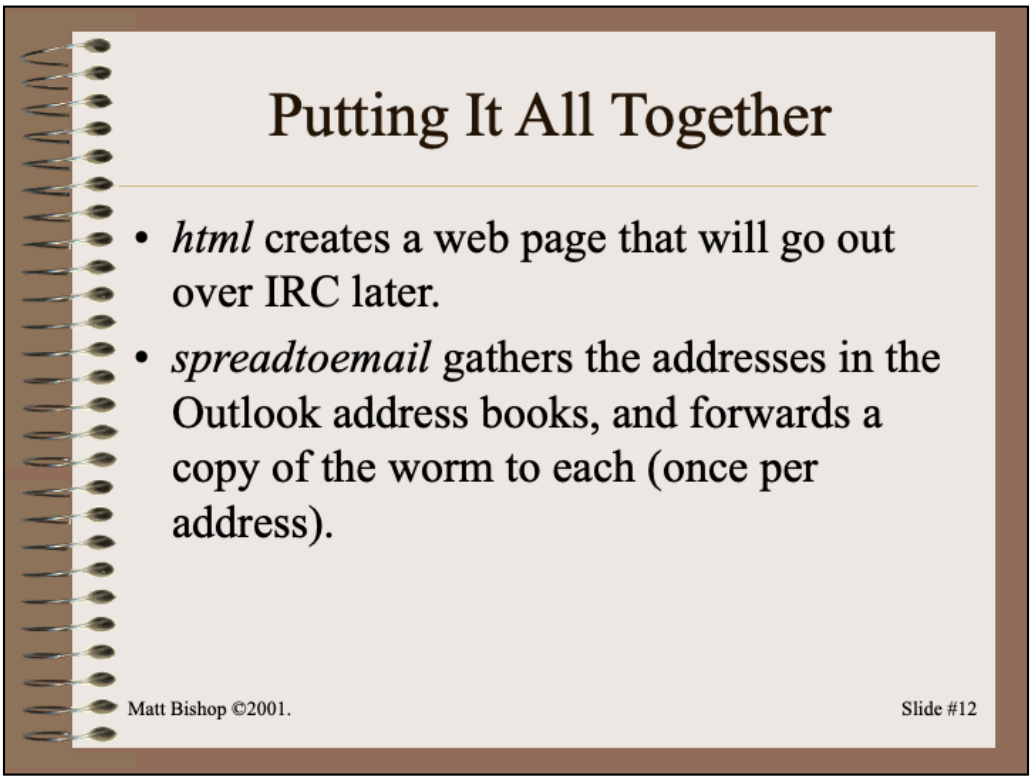

The files that indicate mIRC is being run are mirc32.exe, mlink32.exe, mirc.ini, script.ini, or mirc.help. The script uses the dcc command to send the file.




*main* initializes everything for later use.

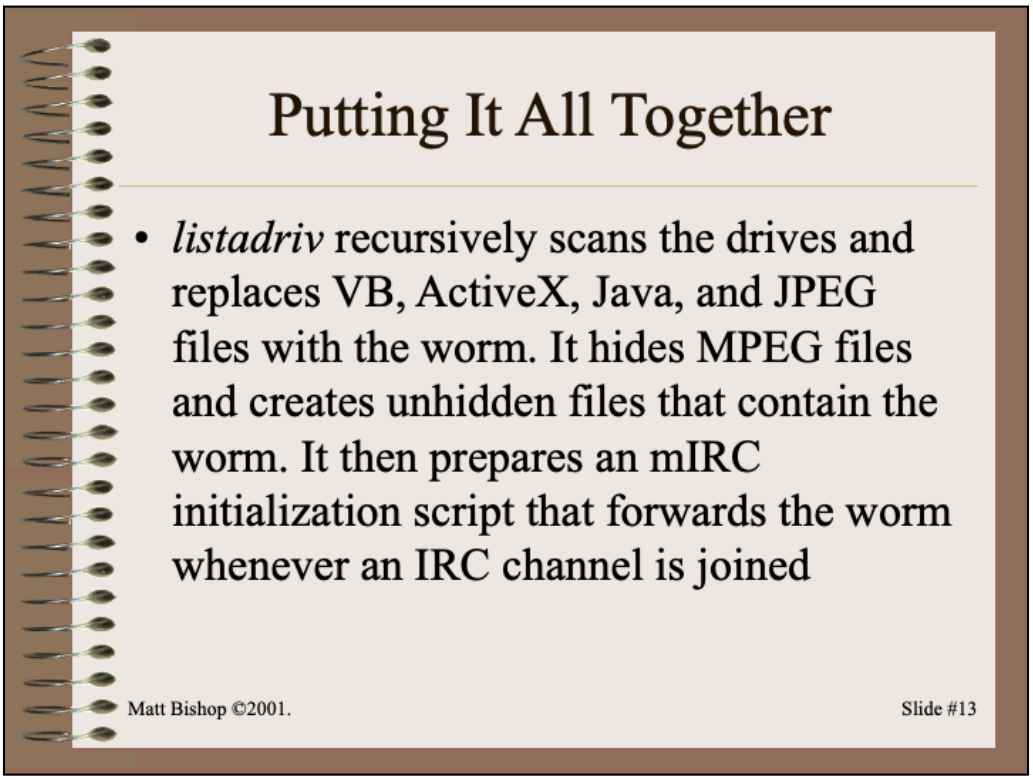
*regruns* arranges for a program to be downloaded at a later time. After the program is downloaded, on the *next* run of the worm (for example, after the next reboot), the downloaded program is added to the set of programs that runs when the system boots. The downloaded program relies on WinFAT32.exe; if that is not present, this step is skipped.

Elias Levy (of SecurityFocus) analyzed the binary and concluded that it emailed any cached passwords to MAILME@SUPER.NET.PH.



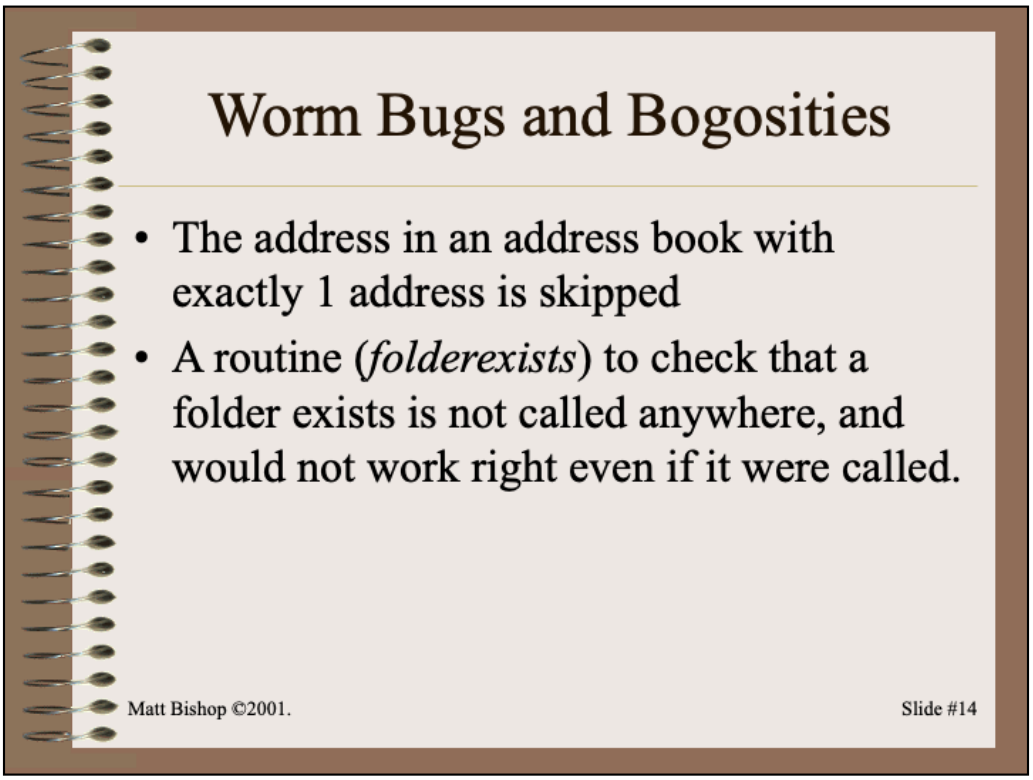

*html* builds the web page as an alternate way of spreading, through IRC.

*spreadtoemail* sends at most one copy per address, but if the same user has multiple addresses, she will get one per address. Also, the worm uses the Registry to track where it has sent the messages, so a new incarnation of the worm will not resend messages to addresses to which they were previously sent (unless the Registry keys are deleted).



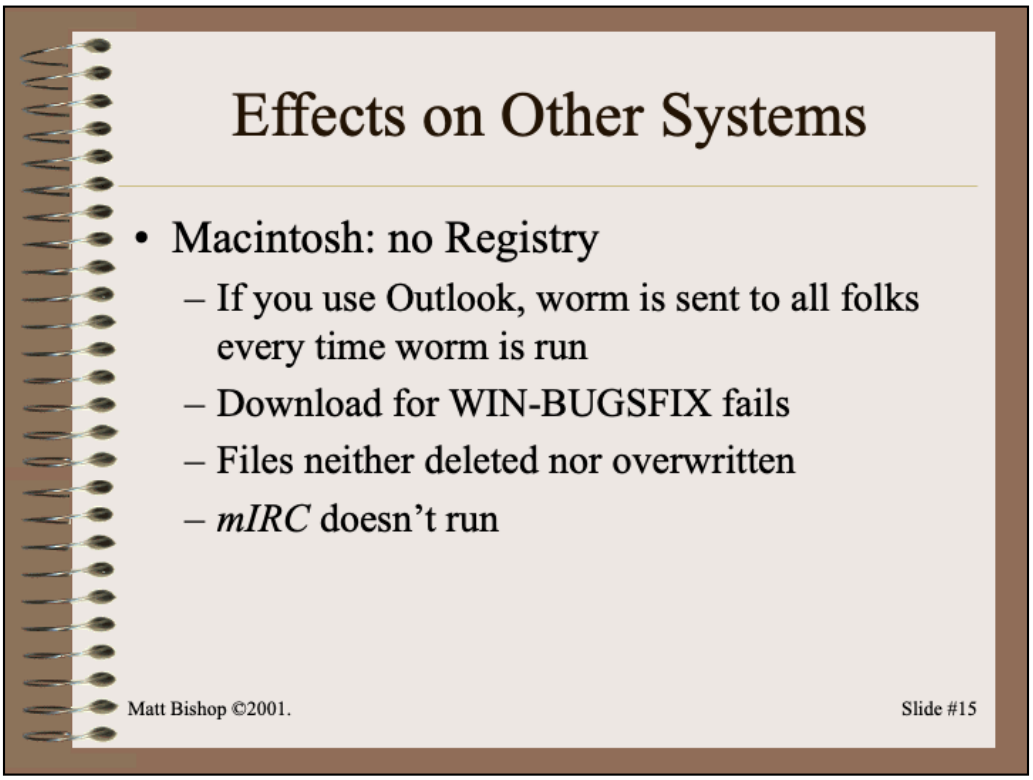

*listadriv* sets up the files so that the user sees the original file name, including extension—if you're not careful and don't notice that the extension is visible, you'll launch the worm when you double-click! (Most people aren't that careful.) Most files are deleted; I'm not sure why the MPEGs aren't, but they are hidden so you don't see them.

The IRC part is particular to mIRC. Other versions of IRC that do *not* use any of the initialization or help files, libraries, or programs won't be affected; but look out; other IRCs also use the script.ini file!

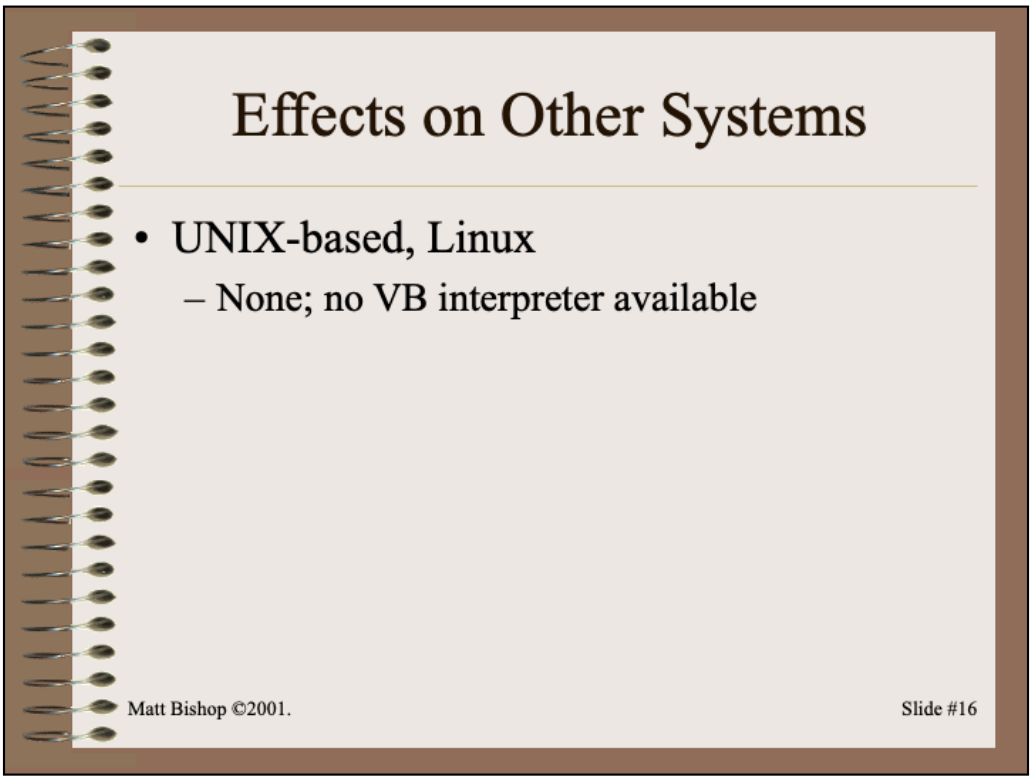



The test for the size of the address list is written so that the current size is compared to the size of the list stored in the Registry. If the Registry key does not exist, the comparison is to 1. This means that when an address list is created, and consists of one user, the comparison (1 > 1) is false and the worm does not propagate. This seems to be accidental (I can't think of why one would do this—all suggestions welcome!), and so I'll call it a bug. Note if the address is in another address book with more than one address, the worm goes to it.

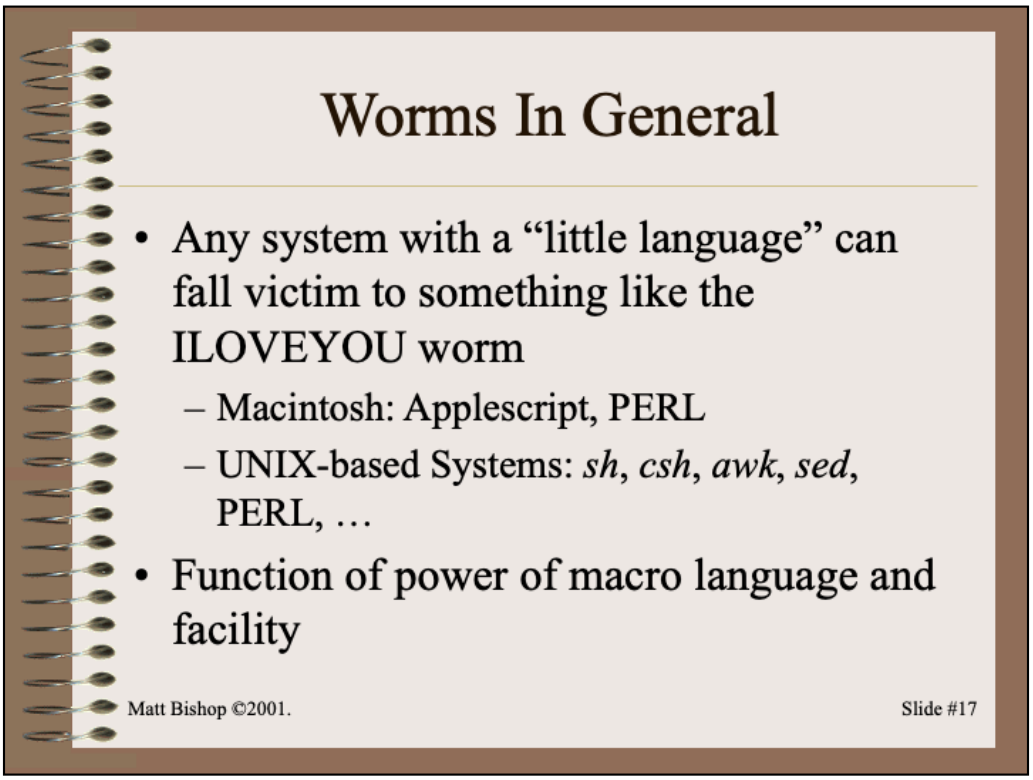
The folder routine invokes a non-existent method and returns result by assigning to *fileexists*—which is the wrong name; it should be *folderexists*.




The worm is annoying but non-destructive on a Macintosh. If you turn off VB or don't use Ourlook, the worm has no effect. If you do, it will create the three copies, but these are never run because they don't go into the Startup folder. Even if the WIN-BUGSFIX file could somehow be downloaded, it wouldn't run. Also, Macintosh IRCs typically do not use a *script.ini* file nor any of the others the worm looks for, so the IRC method of spreading bombs. (mIRC doesn't run on Macintoshes.)

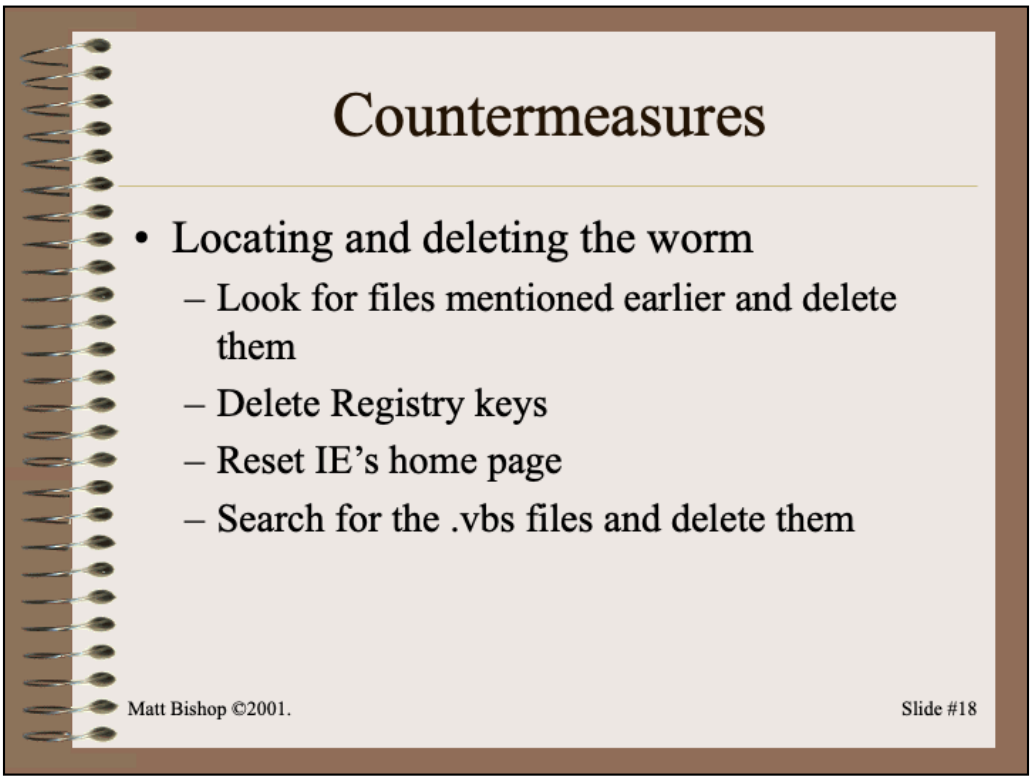



That's how I first saw the worm, as a text file.




A good example of a UNIX virus (which could easily be mutated into a worm) is Tom Duff's paper "Experiences with Viruses on UNIX Systems" in *Computing Surveys* **2**(2) pp. 155–171; it presents a *sh* virus that would run on any UNIX-based system. It's an expansion of a paper he did in the Winter 1989 USENIX Conference called "Viral Attacks on UNIX System Security" (pp. 165–171).

The power of a macro language controls what it can do. If it can access (delete) system resources (such as files) or execute other programs, it can probably be used for malicious purposes.




The files are *rootfolder*\MSKernel32.vbs, *systemfolder*\Win32DLL.vbs, *tempfolder*\WIN-BUGSFIX.EXE*, script.ini in any mIRC folder,* rootfolder*LOVE-LETTER-FOR-YOU.TXT.vbs and* systemfolder\LOVE-LETTER-FOR-YOU.HTM.

The Registry keys to delete are:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\MSKernel32

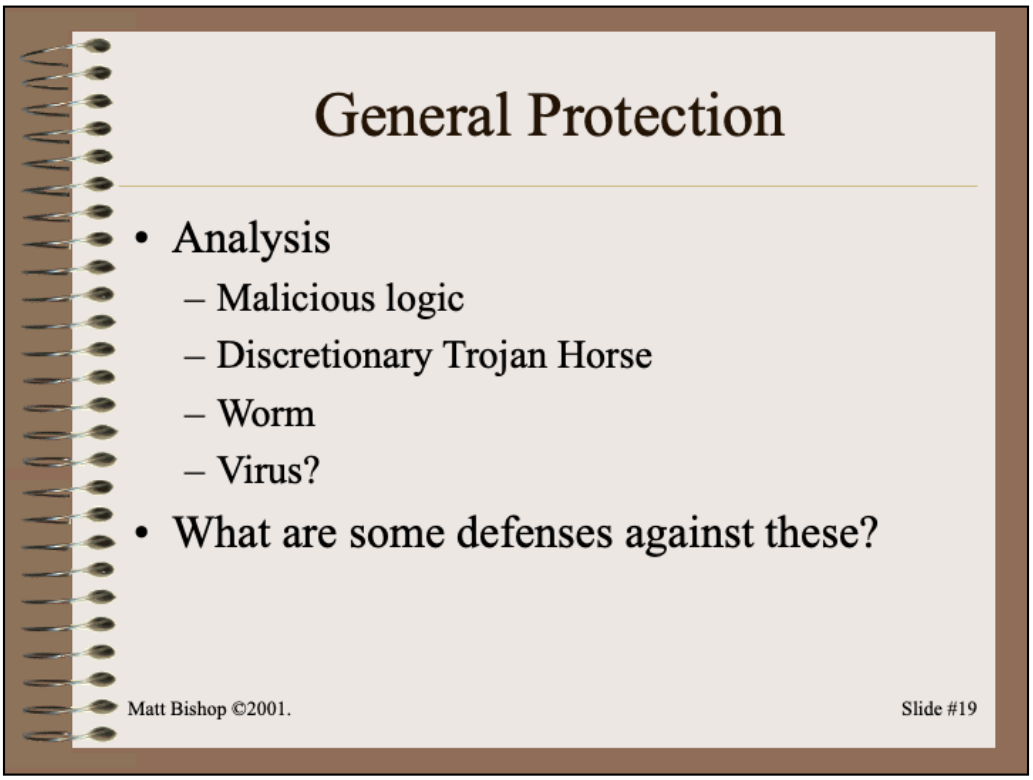
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices\Win32DLL

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\WIN-BUGSFIX

and all of the address keys.

You can reset IE's home page by firing it up and *immediately* clicking on STOP, then go to Tools>Internet Options and change the home page back to what it should be.

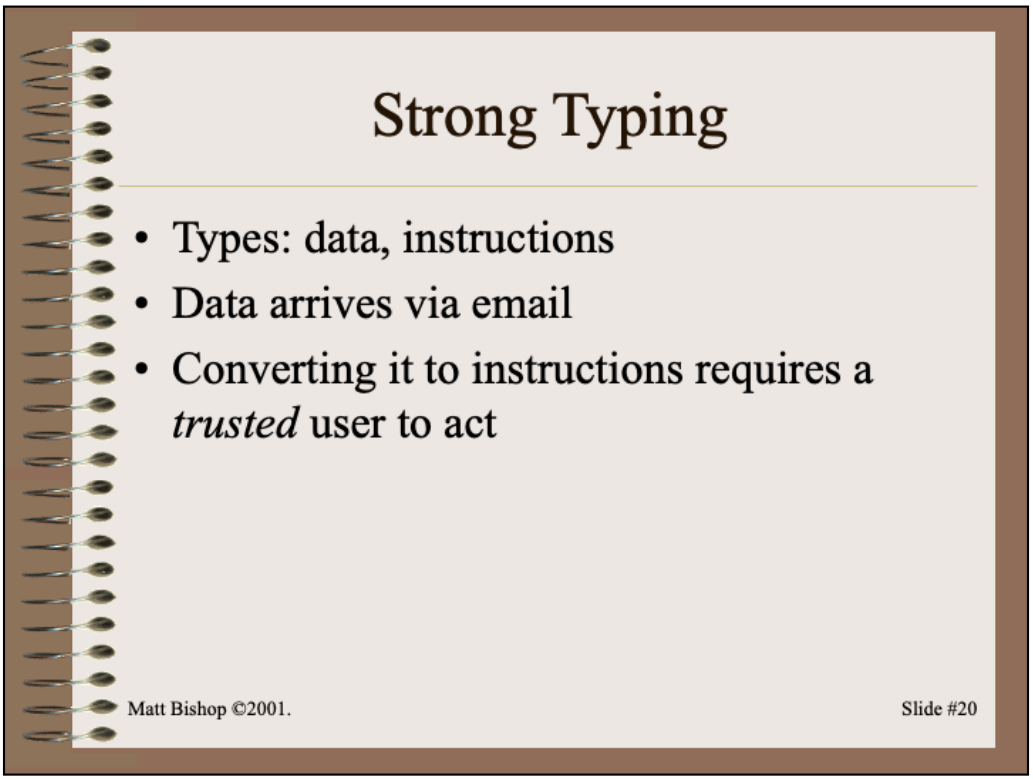
For the .vbs files, look for ones that used to be .vbe, *etc*. Do not execute these; you will reinfect your system!




This worm is a form of malicious logic, defined as "any program or segment of code that acts in violation of the security policy." It's a Trojan horse since it has an overt purpose (to read a love letter) and a covert purpose (see the earlier part of the talk). It's called "discretionary" because it runs with your privileges, and can only do what you can do.

This is the crux of the problem: the ILOVEYOU worm can do only what you can. You could, by hand, do everything the worm does. So, how can security mechanisms determine that you are not doing it? That the worm is doing these things without your knowledge and permission? The inability to answer this question clearly, simply, and *correctly* every time is the reason the Trojan horse problem is a major computer security problem.

ILOVEYOU is also a worm (spreading from machine to machine). Depending on your definition of "computer virus," it may or may not be one (as it doesn't insert itself into any other program or letter, but it does create new letters). The latter is a philosophical, not practical, question though.



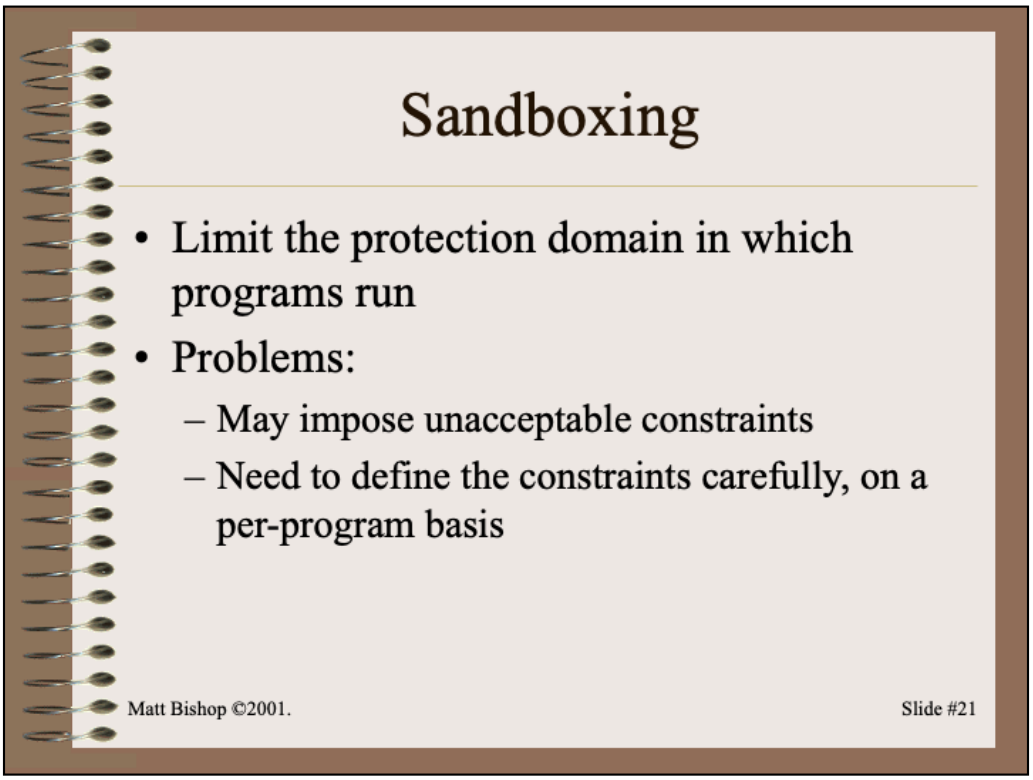

For our purposes, interpreted programs are instructions, even though those programs are fed to an execution engine (the interpreter) and are not executed directly.

Earl Boebert was the first to suggest distinguishing between *data* and *instructions* for the purposes of security. In the system he worked on (Secure Ada Target, later called the LOCK), one had to have a system security officer do the conversion. Older Burroughs machines had a similar distinction, although it wasn't based upon security.

Practical problem: in computing today, this sort of conversion happens *all the time*, for example with email, documents (Microsoft Office macros, as well as other vendors) and downloads from the Web. On most general-purpose systems, this would be considered infeasible. Delegating it to ordinary users leads to the problem we saw here; users had to click on the attachment.

Forbidding *all* executable or interpreted files in attachments or downloads would help, but is probably too draconian for most places. it would not solve the problem, either.

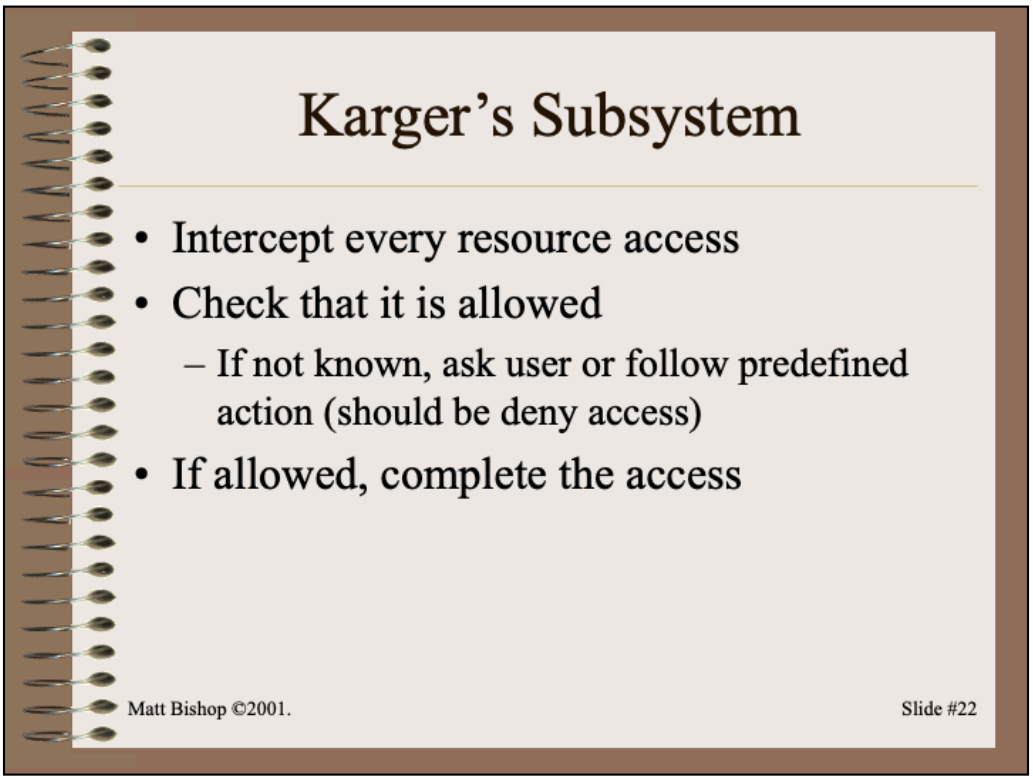
Reference: W. Boebert and C. Ferguson, "A Partial Solution to the Discretionary Trojan Horse Problem," *Proceedings of the Eighth National Computer Security Conference* pp. 245–253 (Sep. 1985).




This is a direct application of the principle of least privilege. The worm can only access a very small number of system resources, so it cannot tie up system resources, it cannot delete anything except those resources it has access to, and it cannot communicate with anyone outside the protection domain.

This is how Netscape handles Java programs. Microsoft is moving in that direction with ActiveX.

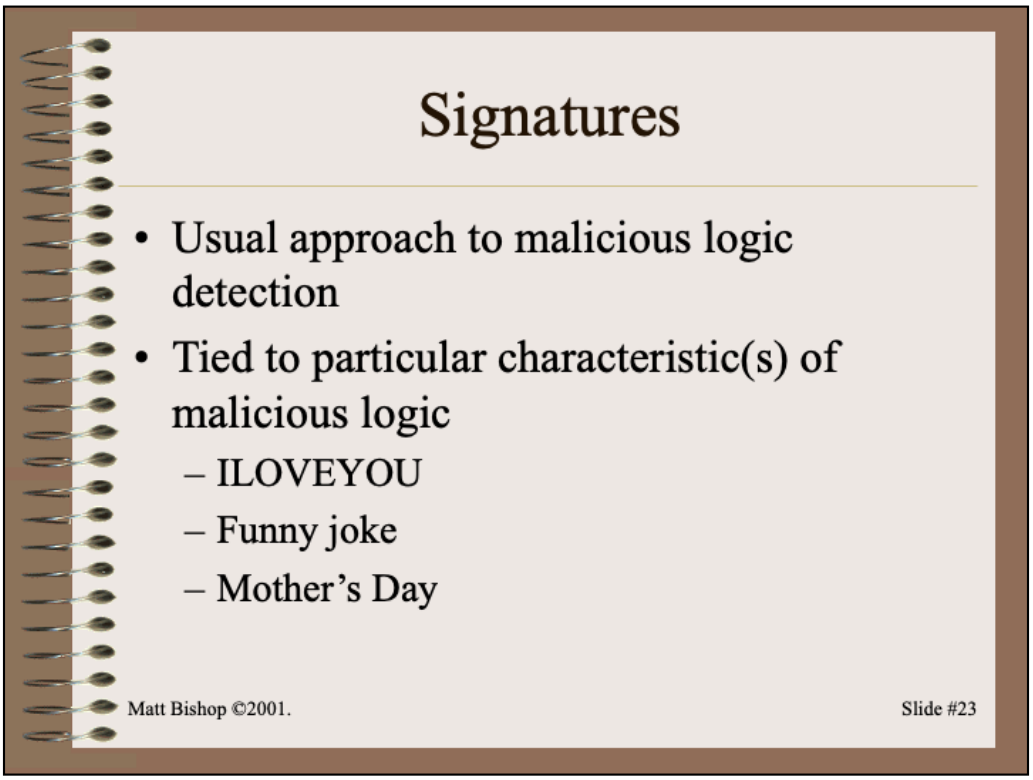
The main problem is defining the constraints tightly enough to prevent any harm while allowing the program to perform any (legitimate) function.



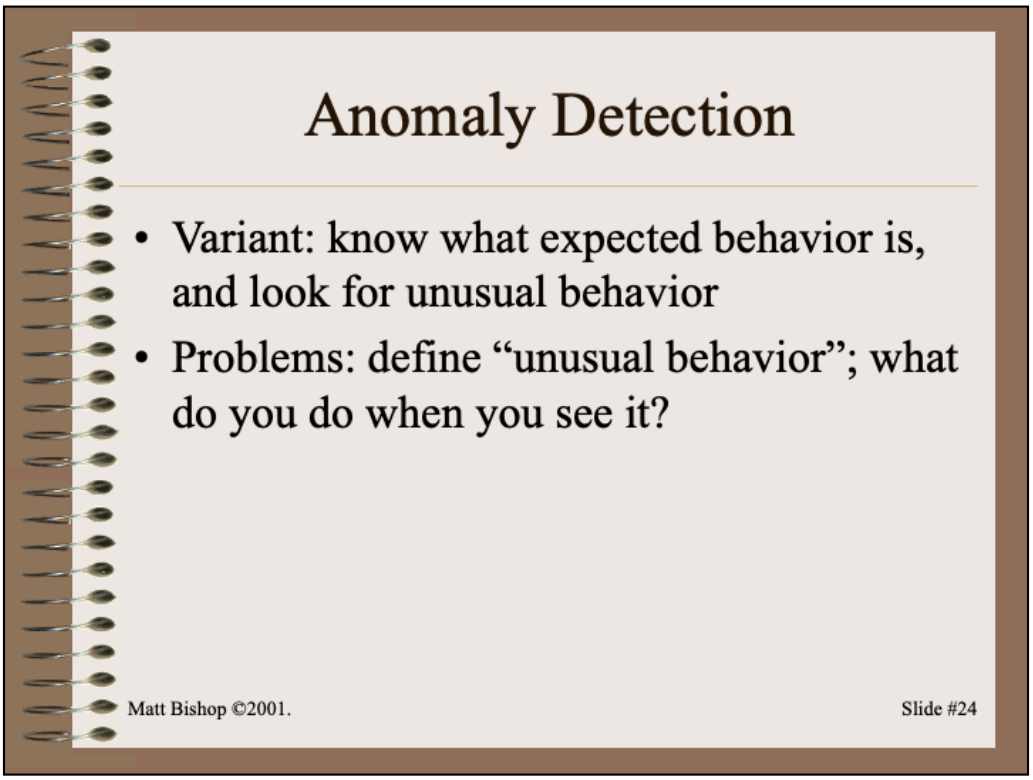

Example: if a C compiler tried to write a file ending in .o, that's fine (it's an object file). if it tried to write to ".login", something's wrong. One of the problems is constructing these tables. The other problem is binding the name in the table to the program (if I rename my C compiler to D, will the subsystem be bright enough to figure out it's still the C program?). Also, on a development system, updating the tables becomes very difficult.

Lai and Gray implemented a version of this on a UNIX system. They found it worked well. Unfortunately, they made several assumptions to minimize the performance impact, such as not including *login*, *sh*, and *mail* in the checking. These vitiated the security benefits, as those programs were the ones attackers tended to hit.

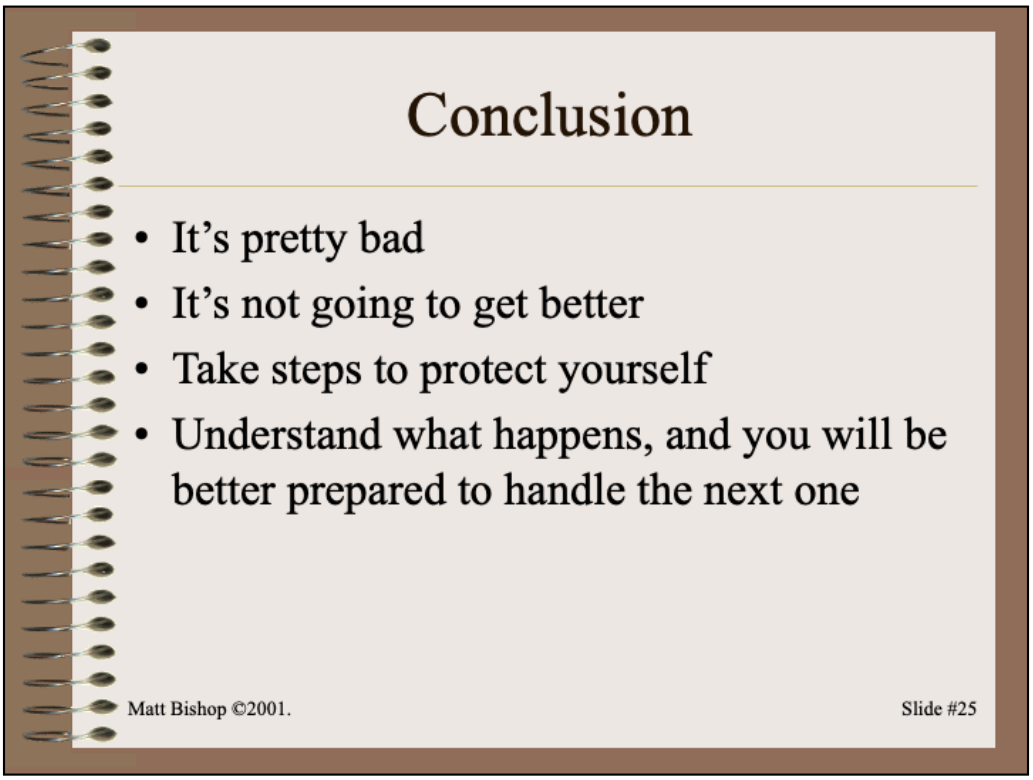
References: P. Karger, "Limiting the Damage Potential of Discretionary Trojan Horses," *Proceedings of the 1987 Symposium on Security and Privacy* pp. 32–37 (Apr. 1987); N. Lai and E. Gray, "Strengthening Discretionary Access Controls to Inhibit Trojan Horses and Computer Viruses," *Proceedings of the Summer 1988 USENIX Conference* pp. 275–286 (Summer 1988).




Vendors released updates to their anti=virus programs that would catch the ILOVEYOU worm. These looked in the Subject field of letters, and flagged any with ILOVEYOU. So, attackers changed the title to "Funny joke". Someone also changed the worm to say that the $300 gift for Mother's Day you ordered had been sent, and would be billed to your credit card account; check the enclosed invoice. The invoice, an attachment, was (of course) the ILOVEYOU worm.




This is the basis for intrusion detection (specifically, anomaly-based intrusion detection). The problem is characterizing unusual (or usual) behavior. In this case, a mail script adding lots of keys to the Registry should have raised eyebrows.




Good luck, everyone! ☺