Report # C4-072R-99
Date: 20 Dec 1999

Microsoft Office 97 Executable Content Security Risks and Countermeasures

Architectures and Applications Division
of the
Systems and Network Attack Center
(SNAC)

Author(s):
Rhonda Breon, C43
Ken Katano, C42

Released By:
Curt Dukes, Chief C43

National Security Agency
ATTN: C43
9800 Savage Rd. STE 6704
Ft. Meade, MD 20755-6704

(410) 854-6191commercial
(410) 854-6510 facsimile

UNCLASSIFIED

Microsoft Office 97 Executable Content Security Risks and Countermeasures

December 20, 1999
Version 1.0

Steven Bonner, Rhonda Breon, Edward Igoe, Ken Katano
Executable Content Technology Team
Systems and Network Attack Center
National Security Agency

ABSTRACT

Office 97 is a popular software package of office applications developed by Microsoft that includes Word, Excel, Access, PowerPoint, and Outlook. Each of these applications includes a programming language for customization of their features.

This paper provides an analysis of each application, including techniques for embedding executable content or mobile code within each application. Each analysis summarizes the executable content threat, provides examples of embedding executable content within each application, and outlines possible countermeasures to protect the user against executable content attacks.

Acknowledgements

The authors would like to thank Neal Ziring for offering his technical expertise and guidance while conducting the research of the Office 97 applications. The authors would also like to thank Don Simard, Mary Kolencik, and Maan Qazzaz for reviewing this document and providing comments that both improved its technical content and readability.

Table of Contents

Microsoft Office 97 Executable Content Security Risks and Countermeasures (U)

Executable Content Technology Team
Systems and Network Attack Center
National Security Agency

1.0 Background

The *Microsoft Office 97* suite includes five separate office applications: *Word* provides word processing capability, *Excel* is a spreadsheet application, *Access* is a database package, *PowerPoint* facilitates the creation of slide shows or presentations, and *Outlook* is a mail/groupware application. *Office 97* runs on Microsoft Windows 95, Windows 98, and Windows NT 3.51 with Service Pack 5 and later versions. Each application features customization capability to satisfy the user's specialized requirements. This customization includes the ability to embed programming instructions within the applications to perform many useful activities. For example, the user can create a button within an Outlook email message that automatically sends responses to a survey back to the sender. However, this customization capability can also be used to perform malicious activities, such as deleting the user's data. Consequently, this paper focuses on the threat potential of embedded code and countermeasures to decrease the threat.

For customization, each Office application includes a development environment. As part of the development environment, the Visual Basic for Applications (VBA) programming language is included in Word, Excel, Access, and PowerPoint. VBA is Microsoft's standard extension language, which is derived from Visual Basic, but designed to execute embedded within other software. VBA is an interpreted programming language complete with features that allow for a multitude of activities, including application control and customization, file manipulation, and system service calls. Visual Basic Scripting Edition (VBScript) is the programming language provided with Outlook. This language only offers a subset of VBA's functionality in that statements that provide file I/O or system service calls were deliberately left out of the core instruction set to make it a "safer" language. However, VBScript in conjunction with the OLE (Object Linking and Embedding) model allows not only for application control and customization, but also the manipulation of objects within Microsoft Object Libraries. Consequently, VBScript within Outlook may be used to manipulate such things as

Outlook mail messages, Word documents, or File objects, thus significantly increasing the application's threat potential.

In addition, each of the Office applications supports ActiveX controls. ActiveX controls are separate binary executable programs which can be written in various programming languages to perform a wide range of activities. All of the Office applications allow the user to insert built-in or customized controls. These controls can then be manipulated by using the included programming language (VBA or VBScript) to write functions or subroutines that respond to a pre-determined set of events. For example, the standard Command Button control responds to several events such as clicking on the button. This type of customization is subject to the security mechanisms in each product. Furthermore, these applications all support HTML format, often known as the language of the Internet. Each application can be converted from its native format to HTML using the S*ave as HTML* option. It is then also possible to include ActiveX controls within the HTML and to script them using a scripting language such as VBScript or Javascript. This type of scripting is then subject to the security mechanisms present in the browser. In addition, it is also possible in Word, Excel, Access, and PowerPoint to insert ActiveX controls as objects. Once again, the security mechanisms vary somewhat depending on the application. In Word, Excel, and PowerPoint, the user will not be warned via the standard macro checker upon opening the container (i.e. document, workbook, or presentation). Rather, a separate dialog about the dangers of OLE is presented to the user with the option to continue if the control is activated.

Using these customization features within the Office 97 applications, an attacker may embed code which allows a wide range of attacks, including exfiltration (i.e. copying data and sending it to another destination), modification, or deletion of the victim's data as well as insertion of programs containing viruses that can be proliferated to other user's machines. Such embedded code executes with the permissions of the victim and often without the victim's knowledge. This concept of delivering code to another user in a format that appears to be passive data, such as a Word document, will be called executable content or mobile code throughout this paper.

The remainder of this document provides a brief overview, the executable content threat, examples, and possible countermeasures for each of the Office 97 applications. There is a separate section for each application which was structured so that individual sections could be read independently without loss of information. These sections were also researched and written by different authors with different writing styles. Consequently, there are variations in the techniques emphasized as well as presentation of the information. It should also be noted that Outlook 97 is currently packaged with Office 97. However, Outlook 98 has been available since the Fall of 1998 and will be emphasized in this paper.

2.0 Description

2.1 Word

2.1.1 Overview

Microsoft Word is the word processing component of the Microsoft Office suite of programs. The widespread availability and ease of use of Microsoft Word has made it a popular target for executable content attacks. There are three main forms of executable content in Microsoft Word. They include VBA macros, ActiveX controls, and scripting with the HTML format.

The primary vehicle for delivery of executable content is VBA. VBA is meant to allow the user to automate complex tasks. However, VBA provides far more capability than required for a simple application extension language. VBA programs are referred to as macros. In Office 97, a macro runs in the host application's process space. This means that Word (or some other Office application) must be running in order to execute a macro. This also means that the macro is limited to the privilege level of the Office user. In a Windows 95/98 environment this affords no protection, but in a Windows NT environment, a user may be restricted from accessing some files or system resources.

In order to run a macro, the document containing the macro must be opened. A macro may be invoked in five ways:

- A macro can be invoked from the Tools menu via the Macro GUI.
- A macro can be triggered by a button in a toolbar.
- A macro can be assigned to a keyboard shortcut sequence. (e.g. Control-M)
- A macro can override a built-in menu selection. For example, a user could define a custom File.Close function which replaces the built-in File.Close function.
- Some macros will execute automatically upon certain events. A macro[1] given the name Document_Open, Document_Close, or Document_New will run when the user opens, closes, or creates a new document respectively. There are also automated macros from older versions of Word that are still supported in Office 97. These are AutoOpen, AutoClose, AutoNew, and AutoExit. These seven macros are dangerous, in that they automatically execute with minimal user intervention. Most macro viruses use this method of invocation.

The second vehicle for executable content in Word documents is ActiveX. While ActiveX controls are primarily associated with HTML (web) pages, they can also be embedded directly into an Office document.

An ActiveX control is a binary object. This means that it has been compiled to run on a specific hardware platform, in a specific operating environment. Thus a control built for an Intel

1. Technically, these three items are not macros, but "document objects". Macros can be (and by default are) stored in the primary template (usually Normal.dot). Document Objects can only be stored as part of the document.

x86 compatible system running Windows will not run on a DEC Alpha system running Windows. Because it is a binary object, it presents the same danger as running any other unknown or untrusted executable object.

An ActiveX control is typically a button or other GUI object, along with its associated functionality. Such controls are usually invoked by mouse-driven actions, e.g. clicks and double clicks. Microsoft distributes a number of such controls, packaged with popular applications such as Office 97, Internet Explorer, and Outlook.

The third vehicle for executable content is via HTML documents (aka web pages). Thanks to OLE automation, Word 97 has a built-in, fully functional version of Internet Explorer. Thus, if a web page is opened with Word, it is subject to all the executable content concerns that Internet Explorer is subject to, including scripting attacks (VBScript and JavaScript), Java Applets, and ActiveX attacks.

2.1.2 Threat Potential

2.1.2.1 Dissemination

Macros are stored as source code, either within the document itself, or within the document's template. In Word, a template is a special document which may contain configuration and customization data for Word documents. Every Word document inherits its properties from at least one template. The default template is the "Normal.dot" template common to every Word environment.

Word macros are spread by disseminating infected Word documents or Word documents associated with infected Word templates. Documents are most commonly shared via email attachments or by shared physical media (floppy disks or shared network drives), but they can also be shared via HTTP. A Word document can be the target of a hyperlink on a web page; activating such a link in Internet Explorer will automatically launch the Word program and open the document.

Word templates need not be co-located with its documents. Word provides the facility to access templates across both local networks and the Internet. Furthermore, the built- in Macro Checker (see Figure 2.1.a) will not detect macros contained in a template, no matter where it is located, unless the latest Microsoft patches for Word have been installed.

The code for an ActiveX control is not carried within a document. Instead, a reference number called a CLSID is embedded into the document. The operating system uses this number to locate and run the actual code for the control. If the control is currently installed on the system, it will run automatically. Pre-installed controls are a concern; there are several known vulnerabilities associated with controls distributed by Microsoft (see section 2.1.3).

2.1.2.2 Invocation

A malicious macro must be invoked to cause its damage. Typically, macro viruses are attached to the Open event and thus will execute automatically when the document is opened. If an event is not used as the trigger, the user must be tricked into invoking the macro. This could be done by attaching the code to a frequently used keystroke combination or menu command.

ActiveX controls are typically used within web pages, but references to controls can also be embedded into Office documents. It is not necessary for the user to explicitly invoke a control; any malicious action can be built into the initialization code, which executes as the control is instantiated. Consequently, it is possible to automatically invoke a control with malicious code when the containing document is opened.

2.1.2.3 Capabilities

The power of VBA running in a Word macro is immense. A Word macro runs with the privileges of the current user. This is essentially the only restriction on the capability of a macro. VBA has File I/O and can invoke WinAPI system calls; therefore, a macro can read or modify any file, and has the capability of exfiltrating information through a variety of means.

ActiveX has even more capability than Word macros. VBA programs cannot directly access the Windows system kernel, but a native executable such as an ActiveX control can. In addition, ActiveX controls can be developed using a variety of programming languages with an extensive range of capabilities, including file manipulation, access to configuration settings, and execution of external programs. Once again, the primary restriction is that the control will only have the privileges of the current user.

2.1.2.4 Ease of Use

Word macros are very easy to create. Word comes with a sophisticated built-in programming environment for creating macros. As VBA is an interpreted language, macros are stored as source code, thus existing macros are easy to duplicate and modify.

In contrast, ActiveX controls generally require some expertise to create. In addition, they are transmitted in binary object code, so they are very difficult to modify.

2.1.3 Example(s)

The first well known example of a Word Macro Virus was the Concept virus. This macro was allegedly written at Microsoft as a proof-of-concept demonstration. It escaped when infected documents were accidentally released on CDs produced by Microsoft. Originally, this was a benign virus - it simply copied itself into other Word documents on the system. Malicious variants have been discovered.

The most infamous outbreak is the Melissa virus. This virus was delivered as a macro within an email attachment. This macro was insidious because it used the victims' address book to mail itself to other victims. These secondary victims were then likely to open the attachment and activate the macro, because the mail message originated from a known (and presumably trusted) acquaintance. Because this virus could actively mail itself, as well as passively wait for the user to share infected documents, this virus spread very quickly, to the point of disrupting some mail servers.

There are two important points to remember about the Melissa virus. First, it could have easily been prevented by the built-in macro checker. Every victim affected either actively enabled the macros, or had previously turned off the macro checker. Second, because a macro executes with the privileges of the Word user, there is nothing to prevent the outgoing mail mes-

sages from “forging” a signature of the current victim. Thus, a digital signature alone does not guarantee the safety of the contents.

Currently, there are no widely known examples of ActiveX attacks embedded in Word documents. There are no technological barriers to the creation of malicious controls; it is just a matter of time before such an outbreak occurs.

Today, the primary danger of ActiveX is not that a malicious control could infect a system, but that a commercially distributed control could be abused. A recent example is the “scriptlet.typelib” control, which was distributed with Internet Explorer version 5. Abuse of this control could lead to the creation of files and the execution of arbitrary code. Microsoft has issued a patch to correct this particular vulnerability, but unpatched systems remain vulnerable, and there is no reason to believe that future controls will be bug free.

2.1.4 Countermeasures

There are several countermeasures to executable content attacks in Word. These generally work equally well against Macros and ActiveX attacks.

- Use a Word Viewer. There are a number of programs (including one available from Microsoft) which will open a Word document without activating any of the advanced features. There are two downsides to this approach. First, the advanced features are not available with a viewer. Second, documents cannot be edited since viewers are read-only tools.
- Take heed of Word’s built-in macro checker as shown in Figure 2.1.a. After macro viruses became widespread, Microsoft developed a macro detection capability for Word. With this activated, if a document contains any “macros or customizations”, the warning dialog box will appear. The document can then be opened with macros enabled or disabled, or the process can be aborted. There are some drawbacks to this approach. First, there can be false-positive alerts. If a document had macros which were subsequently removed, the document will still generate a warning. A macro warning dialog is also generated for non-macro related “customizations” - for instance alterations to the toolbars, or the addition of ActiveX controls. (The standard macro dialog is not triggered if the ActiveX control is inserted as an object. In this case, ActiveX controls which respond to activation cause a warning about the dangers of OLE if the user attempts to activate the control.) Second, when a document is opened with macros disabled, it is opened as a read-only document; it cannot be edited[1]. If the macro checker is disabled, it should be re-enabled (*Tools->Options*; *General* tab, *Macro virus protection* box).
- Use third party protection software. Many popular virus checking applications will scan Word documents for the presence of known macro viruses. While this approach has been moderately successful for “normal” viruses, it will be less successful against macro viruses, because macro viruses are more easily modified. Relatively few commercial products offer protection from ActiveX controls, and most of these are web browser ori-

1. In fact, if changes are made to the document, it can be saved under a new name, but the original will remain intact.

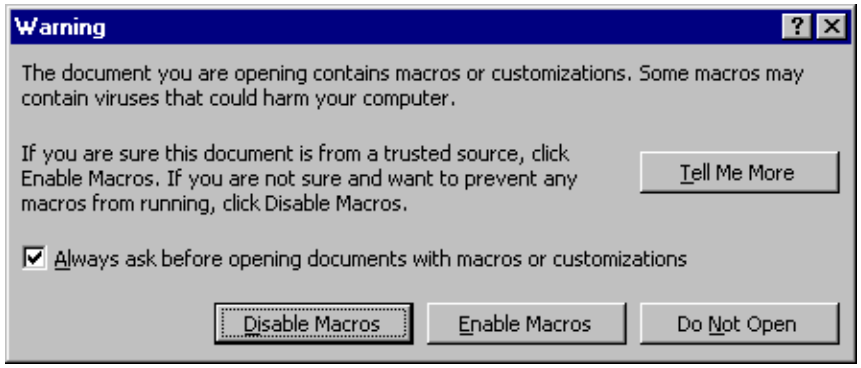


Figure 2.1.a: Word's Macro Checker Warning dialog

ented. It is unclear whether these security products could offer protection from controls embedded in Word documents.

- Don't use Word at all. While this obviously eliminates the threat of Office based attacks, there are two problems. First, it is often impractical to refuse to accept Word documents. They are pervasive, and often the only format in which the desired information is available. Second, other word processing packages are not necessarily safer than Word. In general, this is not a viable option.
- Only open digitally signed Word documents received from trusted individuals via trusted paths. This is Microsoft's preferred security solution. While this can guarantee the source of the document, it does not guarantee that the trusted source was free of infection when the document was sent.
- If an ActiveX control or a hyperlink is encountered within a Word document saved in HTML format, the Word program will apply the security criteria from Internet Explorer before running the control or executing the link. Therefore, it is important to properly configure Internet Explorer, even if using a different product (i.e. Netscape Navigator) for web browsing. This typically translates to enforcing the High security setting for all security zones, or customizing the settings to limit ActiveX as much as possible by either turning them off or forcing the user to respond to warning prompts.
- In addition, it is critically important to have the latest version of Office, Windows, and Internet Explorer, and to install all security patches from Microsoft. The patches and service packs released by Microsoft will correct serious flaws contained in earlier versions of the software.

2.1.5 Summary of Word

Macro viruses pose a serious threat to Microsoft Office users. The best defense is to be alert to the danger, and to trust no document that was externally created.

ActiveX is powerful as an attack vehicle. Avoid running ActiveX controls from untrusted sources. Since it is difficult to detect embedded ActiveX controls, the best protection is to configure Internet Explorer to disable all ActiveX capability.

2.2 Excel

2.2.1 Overview

Microsoft Excel is the spreadsheet component of Microsoft Office. It is capable of all the mainstream spreadsheet functions including organizing data in tabular formats, performing calculations ranging from simple to extremely complex, and providing intermediate as well as final results. It allows the user to organize, sort, format, and print data as well as:

1. save the spreadsheet as an HTML document for incorporation into a website.
2. create and embed hyperlinks within spreadsheets to invoke a web browser and jump to a website, file, or FTP location with a single click.
3. create Web forms, powerful tools which help with gathering input from other Microsoft Excel users visiting a Web site.
4. facilitate user-programmed added functionality, which can be distributed outside the application.
5. create stored templates to pre-format spreadsheets for specified tasks.

The basic layout of the product is best illustrated in the following diagram:

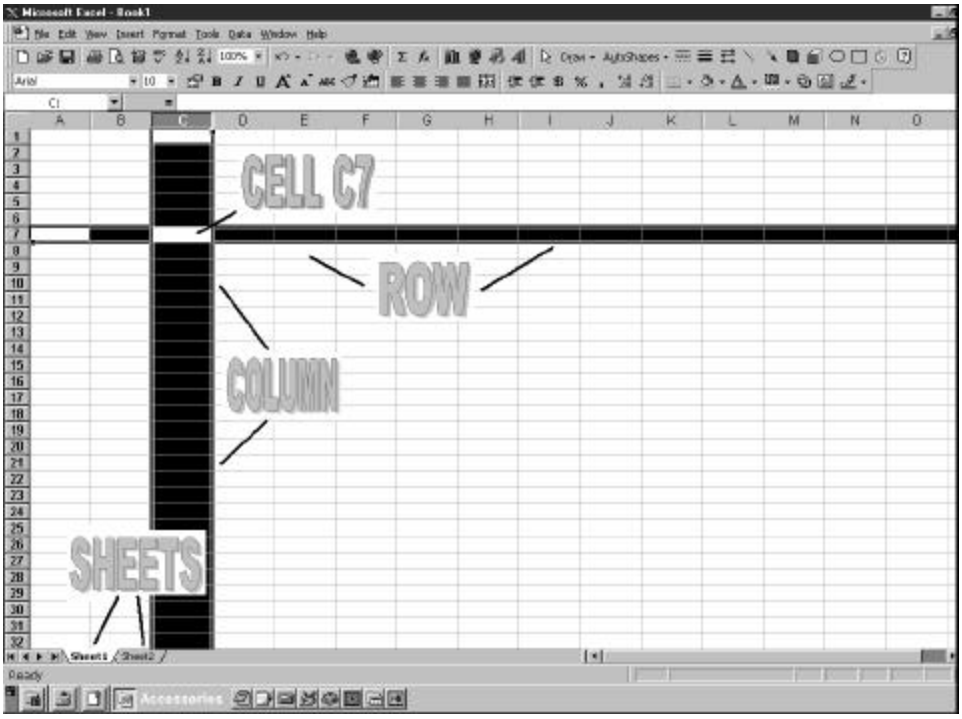


Figure 2.2.a: Excel Worksheet

The *Sheet* tabs, as shown in the lower left corner of Figure 2.2.a, determine the sheet which is currently viewed in an Excel workbook. Each sheet is initially identical, and any number of sheets may exist in one workbook. Each sheet is broken into columns and rows. Each intersection of a column and row is called a cell. Data is generally entered in a cell.

Excel was the first product to support VBA. Excel also supports its own object library for controlling Excel's elements, such as Worksheets and Cells. In addition, Excel includes its own simple formula language and support for ActiveX controls.

Excel's Object Library contains routines and properties for manipulating and accessing Excel's functionality. In Excel, an object represents an element of the application, such as a worksheet, a cell, a chart, a form, or a report. For example, using the delete method of the Worksheet object, an entire worksheet can be deleted through code. In addition, Excel can take advantage of other installed Microsoft object libraries, including those that come in other Office 97 applications. The sharing of these libraries allow programmers a great deal of capability. For example, VBA code within an Excel worksheet may be used to open a Word document, modify its contents, and mail it to another user using Outlook.

Excel formula language includes functions that can be accessed within worksheets to perform tasks for the user. These functions may be used to manipulate values for cells within worksheets directly or they may be called from VBA macros. For the most part, this formula language offers little threat potential since it is primarily used to calculate values for individual cells. However, a vulnerability was found in the Internet community that used the *Call* statement which will be discussed further in the threat section.

As is the case with all of the Office products, ActiveX controls may be included with Excel applications. ActiveX controls are separately compiled programs which may be embedded into an Excel application and controlled via scripts that respond to a set of events. Some controls, such as user interface elements available in forms and worksheets, are built-in. But customized controls may also be included.

Microsoft Excel macros containing VBA and ActiveX controls can be invoked using one of several methods:

- using the TOOLS menu in the open application
- clicking on a custom button attached to the toolbar
- using a custom keyboard sequence
- using hidden re-direction of a standard toolbar selection
- clicking on a hotspot (text, image, ... that activates code) within a spreadsheet
- clicking on a button within a web form
- opening a template containing a macro
- inserting a macro within a workbook event

Workbook events correspond to the following actions:

EVENT NAME	OCCURS:
AddinUninstall	When a workbook is uninstalled as an add-in.
BeforeClose	Before a workbook closes. If the workbook has been changed, this event occurs before the user is asked to save changes.
BeforePrint	Before a workbook (or anything in it) is printed.
BeforeSave	Before a workbook is saved.
Deactivate	When a chart, worksheet, or workbook is deactivated.
NewSheet	When a new sheet is created in the workbook.
Open	When a workbook is opened.
SheetActivate	When any sheet is activated.
SheetBeforeDoubleClick	When any worksheet is double-clicked, before the default double-click action.
SheetBeforeRightClick	When any worksheet is right-clicked, before the default right-click action.
SheetCalculate	After any worksheet is recalculated or after any changed data is plotted on a chart.
SheetChange	When cells in any worksheet are changed by the user or by an external link.
SheetDeactivate	When any sheet is deactivated.
SheetSelectionChange	When the selection changes on any worksheet (doesn't occur if the selection is on a chart sheet).
WindowActivate	When any workbook window is activated.
WindowDeactivate	When any workbook window is deactivated.
WindowResize	When any workbook window is resized.

Figure 2.2.b: Workbook Events

Any of the above events can trigger a macro and its underlying VBA code. The remainder of this section will describe the threat potential of this capability, examples, and possible countermeasures to protect the user from attacks.

2.2.2 Threat Potential

Microsoft Excel macros, written in VBA, have access to almost all other Microsoft Office capabilities, including access to the machine's file system. VBA also includes a SHELL command, which will execute outside executables within Excel's memory space on the computer. The possibilities for exploitation of such a powerful tool are only limited by the hacker's imagination.

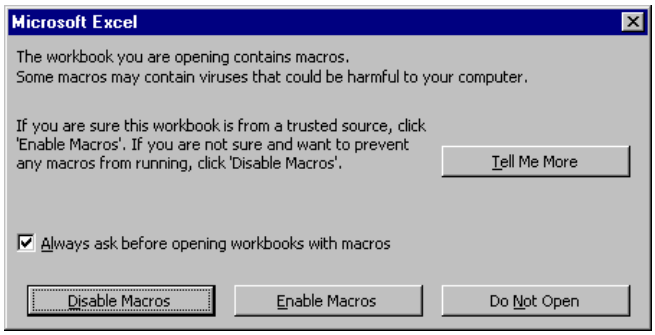


Figure 2.2.c: Macro Warning Dialog

In an attempt to invoke a level of security, Microsoft incorporated a macro checker for workbook files to warn users of enclosed macros before they're opened. When enabled, the macro checker displays the warning box, as shown in Figure 2.2.c, when a workbook containing a macro is opened. If the user clicks *Enable Macros*, the workbook is opened, and the macros are enabled. If the macro is triggered by an event, like the opening of the workbook, malicious code can be initiated. Also note the checkbox on the warning dialog. If unchecked, the macro checker is not invoked and is not enabled again until the user explicitly re-enables it (*Tools->Options* menu; *General* Tab; *Macro virus protection* box). The deficiency in this system is demonstrated by the recent proliferation of an Excel virus named

Papa, which could not be distributed unless users ignored the warning and enabled the macros.

Microsoft also allows a programmer to create and incorporate custom added functionality to Excel in the form of compiled VBA. This is what Microsoft calls an Excel "Add-In". Excel Add-Ins are created by writing and testing the VBA code in the VBA editing environment, compiling the code, and then saving it as an Add-In. These Add-Ins are then moved to a startup directory on the machine and enabled from within Excel. Once enabled, they are opened every time Excel is started, and can therefore be activated based on user actions. Since an Add-In is an extension to Excel, the loading of an Add-In does not pass through the macro checker. Microsoft does not require Add-Ins to be registered like other external components, so a malicious Add-In can be loaded on a machine using the name of an established, benign component. This fools the Excel application into loading the malicious Add-In and enabling it.

The formula language, used primarily within Excel to calculate values for cells, also has threat potential as demonstrated by an alert sent to the Internet community in the spring of 1999. The *Call* function can be used within macros or as a worksheet function to call procedures from dynamic link libraries (DLLs) which are external to a worksheet. If the *Call* function is used as a worksheet function, then the user is not warned. (If the *Call* is invoked from a macro, then the user is warned via the standard macro checker.) Consequently, potentially malicious dlls could be invoked without the user's knowledge. This vulnerability was patched by Microsoft in Office 97, Service Release 2 (SR-2), by disabling the *Call* function.

The ActiveX technology provides additional attack capability as it does in all of the Offfice 97 applications. Customized controls are of particular concern since they are binary executables that run with the user's access rights to the machine's resources, and have vast capabilities. ActiveX controls can either be inserted directly into an Excel spreadsheet, or a reference to an ActiveX control can be added to a worksheet in HTML format. If they are added directly to a worksheet, VBA macros may be written to control them. These macros are flagged by the macro checker as long as it is enabled. If the ActiveX control is added to the HTML, then Internet Explorer is automatically triggered when the control is encountered, and the security settings of Internet Explorer apply. It is therefore important to securely configure Internet Explorer.

2.2.3 Examples

The following example demonstrates an Excel VBA macro which posts the familiar "Hello World" message dialog to the user. Since the Workbook_Open event is used, the macro executes each time the default workbook is opened:

```
Private Sub Workbook_Open ()
    MsgBox ("Hello World")
End Sub
```

A more complicated example of VBA's capabilities is shown in Figure 2.2.d. When invoked, this macro will setup the headers across a page with the numbers from 1 to 10, and number each of the first 20 rows. This code demonstrates the use of Excel's Object Library which includes methods and properties for manipulating Excel objects. For example, the

Range("A1").Select statement selects a set of cells with the *Range* object and defines that area when it calls the *Select* method.

CODE	COMMENT
Sub Macro1()	
Range("A1").Select	*Select the first cell*
ActiveCell.FormulaR1C1 = "1"	*Fill it with the number 1*
Range("A2").Select	*Select the A2 cell*
ActiveCell.FormulaR1C1 = "2"	*Fill cell A2 with a "2"*
Range("A1:A2").Select	*Select cells A1 and A2*
Selection.AutoFill Destination:=Range("A1:A20"), _ Type:=xlFillDefault	*Use Excel's auto fill to fill in the whole column*
Range("B1").Select	*Select the B1 cell*
ActiveCell.FormulaR1C1 = "2"	*Fill cell B1 with a "2"*
Range("A1:B1").Select	*Select the first two cells of the "A" row*
Selection.AutoFill Destination:=Range("A1:J1"), _ Type:=xlFillDefault	*Use Excel's auto fill to fill in the whole row*
End Sub	

Figure 2.2.d: Example 2 using Excel's Object Library

To demonstrate VBA's capability to use Object Libraries from other Office applications, the example shown in Figure 2.2.e opens an instance of Microsoft Word, locates the default document directory in the machine's registry, and opens the first document it finds. After the macro

CODE	COMMENT
Public Sub Start_Word()	
Dim WordApp As Object	*Allocate a variable, type-object*
Dim TargetFile, DocDir, Docpath, SaveFileName As String	*Allocate four string type variables*
Set WordApp = CreateObject("Word.application")	*Create an instance of Ms Word*
DocDir = WordApp.Application.System.ProfileString ("Options", _ "DOC-PATH")	*Get the default location where user saves his Word documents*
Docpath = DocDir & "*.doc"	*Append the extension to create a search path*
TargetFile = Dir(DocDir)	*Get the first .DOC file at the search path*
TargetFilePlus = DocDir & TargetFile	*Prepend the path onto the name of the file*
WordApp.Documents.Open FileName:=TargetFilePlus, _ ReadOnly:=False	*Open the target file*
SaveFileName = (Left(TargetFilePlus, Len(TargetFilePlus) - 3)) _ & "eji"	*Rename the file name with the new extension ".EJI"*
WordApp.Application.ActiveDocument.SaveAs _ FileName:=SaveFileName, FileFormat:=wdFormatDocument	*Save the file with the new name*
Set worddoc = CreateObject("Word.document")	*Create a new Word document*
Set myRange = worddoc.Range(Start:=0, End:=0)	*Define the area of the document to edit*
With myRange	*Using the range just defined...*
.Font.Name = "Arial"	*Choose a font*
.Font.Size = 12	*Choose the font size*
.InsertBefore "You have been"	*Type the words "You have been"*
.Font.Size = 24	*Increase the font to 24*
.InsertAfter " BOMBED!"	*Type the word "BOMBED!"*
.InsertParagraphAfter	*Commit the typing*
End With	
WordApp.Application.ActiveDocument.SaveAs _	*Save the new file with the old filename*
FileName:=TargetFilePlus, FileFormat:=wdFormatDocument	
WordApp.Application.Quit (wdDoNotSaveChanges)	*Quit Microsoft Word*
End Sub	

Figure 2.2.e: Example 3 Using Office's Object Libraries

runs, there will be TWO files: the original with a false extension of "eji", and a new file with the original name and extension. Windows marks the file with the type "Microsoft Word Document", showing no indication that this is not the original document.

Although the effects of the above macro are minimal, and easily reversible, it could have easily deleted the file instead of changing the extension, or it could have copied the contents back to Excel and mailed them to any destination. It could have also accomplished these tasks while looping through all the Microsoft Word, Excel, and/or PowerPoint documents. All of this could be accomplished invisibly and automatically.

These examples were developed for illustration purposes, but there are quite a few known viruses aimed specifically at Excel. The first known Excel macro virus was named Laroux.A, which appeared in July 1996. Laroux.A was not destructive, but was self-replicating, and easy to detect. More recently, in March 1999, X97M/PAPA, a virus that uses the Microsoft Outlook mail program for distribution of infected Excel spreadsheets, was discovered.

2.2.4 Countermeasures

Preventing executable content attacks in Excel would require eliminating the execution of embedded code. This would significantly reduce customization capability in Excel. There are, however, several ways to reduce the security risk posed by executable content attacks.

- Ensure the Microsoft macro warning mechanism is enabled, and that users are instructed to disable macros on documents coming from unconfirmed sites. This can be done by ensuring that the *Macro virus protection* option under the *Tools->Options*; *General* tab is checked. The user can also install a tool, written by Ed Igoe of C43, which re-enables this virus protection feature if it has been disabled. This tool may be obtained from the C43 home page (www.c43.c.nsa), under the *resources* link, *ForceMacro Microsoft Excel Add-In.*
- Set the attributes of the directory where Excel Add-Ins are stored to "READ ONLY". This will prevent an advanced user from creating and installing his own Add-Ins, but would also prevent unidentified Add-Ins from being installed.
- Set the attributes of the PERSONAL.XLS file to read-only. This file is the target of many macros including Laroux.A, Laroux.B, and Laroux.C.
- Install all security patches from Microsoft to protect against known attacks.
- Properly configure Internet Explorer, even if using a different product (i.e. Netscape Navigator) for web browsing. This typically translates to enforcing the High security setting for all security zones, or customizing the settings to limit ActiveX as much as possible by either turning them off or forcing the user to respond to warning prompts.
- Use third party protection software. Many popular virus checking applications will scan Excel spreadsheets for the presence of known macro viruses. While this approach has been moderately successful for known viruses, it will be less successful against macro viruses, because macro viruses are more easily modified.

2.2.5 Summary of Excel

Like the other Microsoft Office products, Excel presents a mobile code threat. History has proven that users routinely ignore the macro checker, causing their own misfortune. Commercial virus checkers have not proven efficient at detecting malicious mobile code. Instead of being proactive and searching for code that looks anything like a virus and then warning the user, the most popular virus checkers are reactive, issuing specific checks for specific macros after those macros have a chance to spread out and do their damage. To help secure Excel against executable content attacks, it is important that users implement the countermeasures outlined in the previous section.

2.3 Access

2.3.1 Overview

Microsoft Access is a database package which provides users with the ability to design, populate and query databases within a standard, Microsoft Windows environment. Of concern from an executable content perspective are the programming languages available. Access allows three programming languages:

1. Structure Query Language (SQL, pronounced “sequel”).
2. Access Macro Language
3. Visual Basic for Applications (VBA for Access).

SQL and Access macros were designed primarily to manipulate database records, and do not have the more general-purpose capabilities of VBA (as we shall see later). SQL and Access macros have been around for some time, and pre-date VBA. For this reason, they do not fit readily into an object-oriented model. However, SQL and macro commands can be issued from a VBA program, using the `DoCmd` object. Thus, virtually any command which can be issued in Microsoft Access can be done from within a VBA program.

2.3.2 Threat Potential

Since VBA for Access is an extension of the Basic programming language, it includes commands which go far beyond, and are unrelated to, database queries and updates. Some of these commands are problematic for security reasons, such as those that provide unrestricted file I/O, including deletion of files and creation of new files containing binary data. To make matters worse, VBA has introduced a `shell` command which allows execution of arbitrary executables. For example, a malicious VBA program could contain a call to format the user’s hard disk.

The security vulnerabilities of VBA for Access pose more than just a hypothetical threat. Actual viruses have been written using Access macros, and have been described on the internet. There are three known Access macro viruses, which all operate in the same way--they search for database files (files ending in “.mdb”) and infect them. They are called “AccessIV” (strains A and B) and “TOX:”

- AccessIV strain A is the first known Access Virus. It runs only in Access97, and is written in VBA. It infects only .mdb files in the current directory.

- AccessIV strain B is a newer, “improved” version, which searches in all directories. It is written in the earlier macro language for MS Access 2, so as to infect a wider “gene pool” of databases. AccessIV is also known by the name “JETDB.”
- TOX does the same as AccessIV strain B, except that it tries to conceal its presence by making itself a “hidden file” and removing an Access pull-down menu that allows the user to display such files. Unlike both of the AccessIV strains, the user cannot prevent the automatic loading of the virus by holding down a “bypass” key during startup.

Commercial countermeasures along with an internally developed countermeasure are presented in the Access Countermeasures section.

2.3.3 Examples

Example 1: Issuing a SQL query from a VBA program:

The following example illustrates issuing a SQL command for manipulating an Access database from a VBA program.

```
DoCmd.RunSQL(“DELETE * FROM StudentPersonal IN college.mdb;”)
```

This command deletes all records (*) from the table “StudentPersonal” in the Microsoft database file “college.mdb”. When this command is executed, the user is prompted to confirm whether he really wishes to delete these records. If someone wished to maliciously delete all of these records without a user’s knowledge, he could first issue the following VBA command, which turns off the Access option to confirm deletes:

```
SetOption “Confirm Action Queries”, False
```

Example 2: Issuing an Access macro action from a VBA program:

To delete a database macro called “zed” from a VBA program, we can use the `DeleteObject` macro action:

```
DoCmd.DeleteObject acMacro, “zed”
```

A third example illustrates an internally developed countermeasure which is presented in the next section.

2.3.4 Countermeasures

Access does not provide the macro checker to warn the user about embedded VBA macros. This is a serious limitation that hopefully will be remedied in Office 2000.

TouchStone Software of Huntington Beach, California, sells an anti-virus software product called PC-CILLIN which supposedly detects and cleans the AccessIV virus. The Data Fellows product F-SECURE can detect and clean both AccessIV and TOX. However, products that look for signatures associated with specific viruses are always one step behind the virus creators. A more comprehensive approach would be to scan a database for a macro called “autoexec.” Since Access automatically executes any macro having this name, virus authors “boot” their viruses by invoking them from an autoexec macro. The VBA for Access module

shown in Figure 2.3.a, written by Steve Bonner while in C43, “inoculates” a specified database by replacing the autoexec macro with a harmless macro. After inoculation, the user can load the suspect database into Access and examine it. The original autoexec macro is renamed “suspect,” and can safely be browsed, along with any VBA modules present.

The macro called “harmless” contains only a single macro action, which displays a message box informing the user that the database has been inoculated. This is arbitrary, and can be replaced with any other desired action, or no action at all.

```
Option Compare Database
Option Explicit

Function inoculate()

    ‘ This VBA function inoculates a specified Access database
    ‘ which may potentially contain a virus (such as the
    ‘ known viruses “AccessIV” and “TOX”.) When this
    ‘ function is invoked, it alters the specified database
    ‘ by replacing its “autoexec” macro (if present) with
    ‘ another, harmless macro. The original “autoexec” is
    ‘ copied into a macro called “suspect”, where it can
    ‘ be examined without automatic execution.

    Dim dbname As String
    On Error GoTo leave

    ‘ prompt the user for the name of the database
    dbname = InputBox(“Database to inoculate?”)

    ‘ if no database name given, just exit
    If dbname = ““ Then
        Exit Function
    End If

    ‘ if database name does not contain “.mdb”, then append it
    If InStr(dbname, “.mdb”) = 0
        dbname = dbname & “.mdb”
    End If

    ‘ if the specified database does not exist,
    ‘ display an error message and exit
    If Dir(dbname) = ““ Then
        MsgBox “Database  ““““ & dbname & ““““  not found.”, 16
        GoTo leave
    End If

    ‘ copy the autoexec macro to the current database temporarily
    DoCmd.TransferDatabase acImport, “Microsoft Access”, _
        dbname, acMacro, “autoexec”, “temp”

    ‘ replace the old autoexec with a harmless macro
    DoCmd.TransferDatabase acExport, “Microsoft Access”, _
        dbname, acMacro, “harmless”, “autoexec”

    ‘ place the saved autoexec into the macro “suspect”
    DoCmd.TransferDatabase, acExport, “Microsoft Access”, _
        dbname, acMacro, “temp”, “suspect”

    ‘ delete our temporary copy of the old autoexec
    DoCmd.DeleteObject acMacro, “temp”

    ‘ let the user know that the innoculation is complete
    MsgBox “Database  ““““ & dbname & ““““  inoculated.”, 64

leave:
End Function
```

Figure 2.3.a: VBA Inoculate Macro

2.3.5 Summary of Access

Although malicious code within Microsoft Access databases has been limited so far to relatively harmless viruses (which do nothing except copy their code into other databases), the VBA language allows for severe system compromises. Thus, the viruses have been fairly benign only because the hackers had no interest in causing destruction. This can change at any time. Furthermore, the ease of use of VBA makes it possible for even unsophisticated programmers to write malicious code. Thus, it is recommended that the inoculation module listed above be run against any Access database obtained from an untrusted source.

Possible future research might include:

- Writing firewall or desktop filters to scan and/or inoculate incoming MS Access databases
- Defining a subset of VBA that might be more secure (for example, would it be feasible to restrict I/O to files other than databases?) The `shell` command that executes arbitrary binary files might also be disabled.

2.4 PowerPoint

2.4.1 Overview

PowerPoint 97 is Microsoft's multimedia presentation application within the Office 97 suite of applications. It allows users to create on-screen, automated slide shows which may include not only textual information, but also images, charts, animation, and sound.

As is the case with most of the Office 97 applications, PowerPoint also uses the VBA programming language for customization purposes. PowerPoint's VBA offers not only the features of the Visual Basic programming language, but also extensions to access PowerPoint's specialized features. These extensions are included with PowerPoint's Object Library which includes objects, methods, and properties for manipulation of PowerPoint's elements. In addition, Microsoft's Object Linking and Embedding (OLE) technology provides a means for integrating solutions across other Microsoft applications, including Excel, Word, Access, and Outlook.

Due to the high capability of the VBA language which is included in PowerPoint and the ability to integrate other applications within a PowerPoint presentation, the threat potential from embedded executable content is significant. The following sections describe the various methods for executing programs from PowerPoint, their threat potential, and possible countermeasures.

2.4.2 Threat Potential

The threat potential from embedded executable code within PowerPoint presentations is significant due to the following reasons:

- The programming language included within the product, VBA, contains many capabilities that can threaten users' resources.

- The executable code can be triggered based on user or system interaction without the user's knowledge.
- Presentations or variants thereof (such as Add-Ins or Templates) can be delivered to another user via e-mail or other shared media. In addition, PowerPoint presentations may be shared via the web by selecting a hyperlink on a web page.

VBA is a full-featured programming language which includes file interaction capability, manipulation of registry settings, and the insertion and execution of external programs. Consequently, a VBA program may perform such malicious activities as deleting, modifying, or extracting a user's files; changing a a user's security posture by changing key values within the registry; inserting and executing an external, malicious program. In addition, the PowerPoint Object Library provides methods and properties for manipulating PowerPoint presentations. This may include the extraction, deletion, or modification of entire presentations, selected slides, or elements from a single slide. It is also likely that an attacker would have other Microsoft libraries available as well, since users typically install all of Office 97. Consequently, the object libraries for Word, Excel, Access, and Outlook are likely to provide additional attack avenues. For example, a macro written in PowerPoint could use Outlook's object model to deliver important Word documents to an attacker.

There are several macro activation techniques available within PowerPoint:

•Menu bar
The *Tools->Macro->Macros* menu option brings up a dialog which then lets the user choose to run a specified macro. This option is useful for testing purposes.

•Customized Toolbar
Customized toolbars and buttons can be used to invoke macros. (The *Tools->Customize*; *Toolbars* tab is used to create a new toolbar. Customized toolbar buttons are added by choosing *Commands* tab and *Macros* from the *Categories* window. A macro can then be selected from the *Commands* window and dragged to the toolbar.) Customized toolbars are available whenever the user activates PowerPoint.

•Object Created on a Slide or Master
Objects created on a slide or master can also be used to invoke macros during a presentation. Such objects may include images, Action Buttons, textual data, and ActiveX controls. To assign a macro to any object, the user can use the *Actions Settings* dialog (*Slide Show->Action Settings*). The user is given the choice of having the macros execute when the object is clicked or when the mouse is dragged over. When activated, a macro can be set to run by choosing the *Run Macro* radio button and selecting a macro from the pull-down list.

•Auto_Open Event of a PowerPoint Add-In
PowerPoint differs from most of the other Office 97 applications in that it does not support attaching macros to the New, Open, or Close events on the document. So, a macro cannot be set to execute based on the opening of a presentation. However, macros in PowerPoint Add-Ins can be set to execute automatically on the Auto_Open event. Consequently, macros can be set to execute automatically when the associated Add-In is opened. For more information on this technique, see section 2.4.2.3.

There are several methods for including executable programs within PowerPoint applications. These methods include embedding programs within UserForms, Templates, Add-Ins, Hyperlinks, ActiveX controls, and Action Buttons. Presentations may also be viewed as web pages by using a browser, such as Internet Explorer (IE). In addition, PowerPoint presentations may be packaged with a viewer to give to other users. Consequently, this *Pack and Go* technology was also researched for security concerns. The threat potential for each method of embedding programs and invocation techniques will be described in the following sections.

2.4.2.1 UserForms

UserForms are custom-designed dialog boxes used to retrieve information from the user. UserForms can contain several different types of components or controls to interface with the user, including buttons, textboxes, listboxes, radio buttons, and checkboxes. Event-driven macros may be attached to both the form itself and the various controls. The macros are set to execute based on actions taken by the user. For example, Figure 2.4.a shows a UserForm with two controls: a listbox for listing favorite dogs and a command button, entitled Exit, for closing the form. When the user clicks on the command button, a customized macro written in VBA may be designed to execute. Another macro could execute based on another event, such as the mouse moving over the listbox.

Fig 2.4.a: UserForm to Retrieve Data From User

However, the UserForm's macros are not automatically launched when the user opens the slide show. Rather, the UserForm must be attached to some type of triggering mechanism such as a Toolbar button or Action Setting. Toolbar buttons can be added to the standard PowerPoint toolbar. They are useful for activating code while creating slides (Slide mode). Action Settings include two choices for triggering an action: mouse click on the object or mouse dragged over the object. In response to one of these actions, the designer can run a VBA macro developed within PowerPoint. This macro can then present a UserForm to the user using the *Show* command which initiates the form and any macros associated with the form's Initialize event. Additional macros can then be associated with user interface controls on the form, such as the listbox or Exit button as shown in Figure 2.4.a.

Since UserForms can contain VBA macros, the threat capability is high if these macros are executed. Two factors limit the threat potential of UserForms. First, macros within UserForms are flagged by the macro checker when the containing presentation is opened, assuming the checker has not been disabled by the user. Although the user is still given the option to execute these programs, he is warned that they may be harmful as shown in Figure 2.4.b. Second, VBA code embedded within UserForms cannot be executed immediately upon the opening of the containing presentation. Rather, the user must go through a series of steps, including opening the presentation, enabling the macros, and either activating the appropriate Action Setting or a customized toolbar option. Consequently, there are several actions required by the user in order for embedded macros to execute. For a detailed example of how to attach and run macros in a UserForm, see Appendix A.

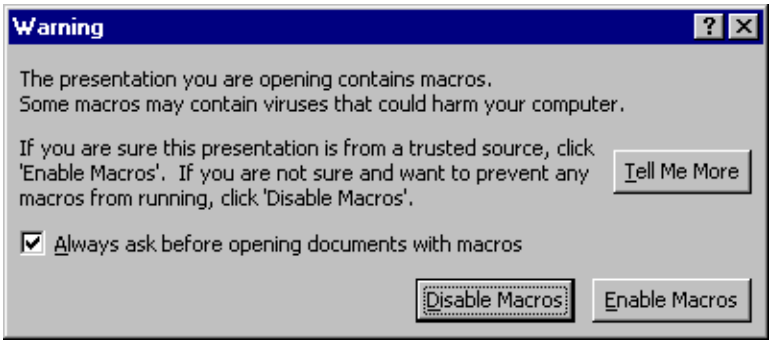


Figure 2.4.b: Macro Warning Dialog

2.4.2.2 Templates

Templates provide a reusable format or model for PowerPoint presentations. Templates may provide a few or many standard features, including background, color scheme, graphical elements, or placeholders for such items as titles, bulleted lists, or charts. Office 97 prepackages several different templates with the PowerPoint application, but it is also possible for the developer to customize their own templates. These templates may include not only the elements mentioned, but also code.

To create a template, the developer can use the *Save As* command from the *File* menu. To save as a template in PowerPoint, the file must be saved as a *.pot* file and the file must be stored in the Microsoft Office Templates folder. Code can then be added to a template using the Visual Basic Editor (*Tools->Macro->Visual Basic Editor*).

The macro checker successfully warns the user about macros within templates, regardless of how the templates are opened, as long as the checker is enabled. In other words, the template with embedded macros can be opened as a template, as a presentation using the template, or through Internet Explorer. All cases trigger the warning dialog to the user as shown as in Figure 2.4.b. However, the user can easily disable the macro checker by unchecking the checkbox entitled, *Always ask before opening documents with macros.* It is therefore very important to educate users on the importance of maintaining secure configuration settings to protect against executable content attacks.

2.4.2.3 Add-Ins

One of the most dangerous vehicles for executable code in PowerPoint is Add-Ins. Add-Ins are reusable packages typically used to deliver a functional graphical user interface. Add-Ins are created by using the *Save As* command from the *File* menu and choosing the .ppa file type. Like templates, Add-Ins may include VBA code. Unlike templates or presentations, that code may be triggered when the user opens the Add-In by using the Auto_Open event. Once Add-Ins are saved, the user can no longer modify or even view the code. Add-Ins must be loaded before they can execute. This can be done from the *Tools->Add-Ins* menu or programmatically for a session using the *AutoLoad* or *Loaded* properties of the *AddIn* object.

Once an Add-In is loaded, code triggered by the Auto_Open event is executed automatically when PowerPoint is started. **This code executes without warning to the user every time the user opens PowerPoint**. Add-Ins may also be shared with other users, for example, as e-mail attachments. When the user opens the attachment, the attachment's file extension signifies

that the PowerPoint application should be started. PowerPoint then issues a warning, as shown in Figure 2.4.b, that the Add-In contains macros which may be harmful to the user's system, assuming that the user has not turned off their macro checker. If the user enables the macros, then the embedded VBA code may threaten the user's data every time PowerPoint is opened without further warning to the user. Therefore, it is very important that users avoid loading or accepting Add-Ins from untrusted sources.

2.4.2.4 Hyperlinks

Hyperlinks allow movement from one location to another when selected within a presentation. For example, hyperlinks may be used to display and execute standard Internet content, allow the execution of another program on the user's machine, or allow movement between Office applications, such as Word and Excel. Hyperlinks to URLs are also maintained if the user converts the presentation to HTML (*File->Save as HTML)*. These HTML files may then be shared over the Internet and viewed using a browser such as Internet Explorer. Warnings to the user about active content within the referenced location vary from non-existent to extensive depending on the application activated when the hyperlink is selected and the user's configuration settings.

Any text or visual object can by a hyperlink, including Action Buttons, clip art, and charts. There are two basic methods for adding hyperlinks to a presentation which include inserting the hyperlink into a presentation or applying Actions Settings. To implement the first method, the hyperlink must be associated with an object by creating an object and then selecting it. The user then chooses the *Insert->Hyperlink* menu option (or the appropriate toolbar button or key sequence). The user must then enter a URL or file location as shown in Figure 2.4.c. The hyperlink is activated, along with associated code, when the user clicks on the object while viewing the presentation in Slide Show mode. The second method for including hyperlinks in

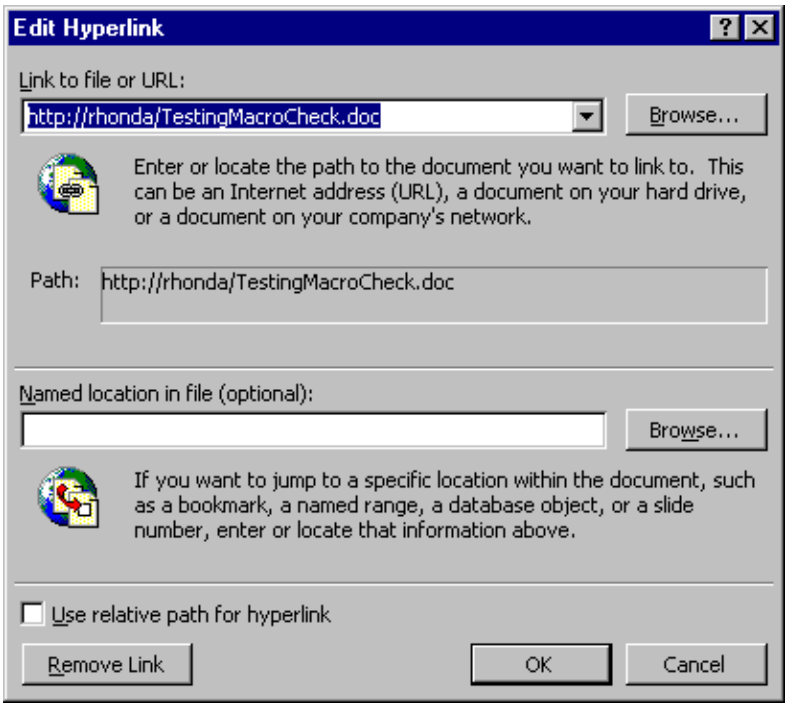


Figure 2.4.c: Inserting Hyperlink Into Presentation

a presentation is to use Action Settings. Action Settings may be applied to textual information as well as a visual object such as an image or Action Button. Action Buttons are buttons added to presentations to perform some service, such as proceeding to the next or previous slide, playing a sound, starting an application, or linking to a web page on the Internet. They are easily inserted onto a slide or the master (*Slide Show->Action Buttons)*.

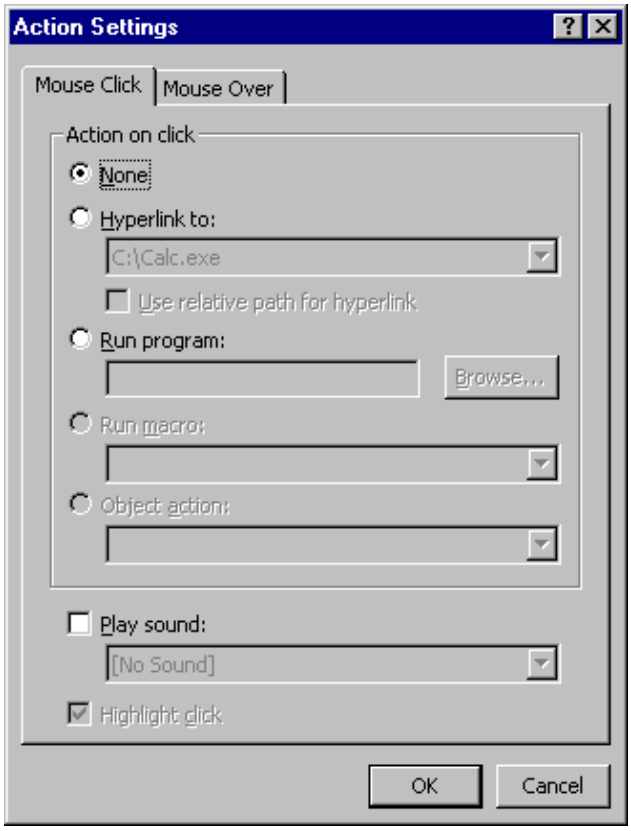


Figure 2.4.d: Action Settings

The *Action Settings* dialog then offers several options for Action Buttons or any other selected object or text. To insert a hyperlink, the developer must choose the *Hyperlink to:* radio button as shown in Figure 2.4.d. The developer is then presented with another list of choices for the hyperlink including transferring control to another slide, PowerPoint presentation, application, or URL. In each case, the developer may choose between two events to activate the hyperlink: mouse click on or mouse drag over the associated graphic.

Since hyperlinks may reference executable code, they pose a security risk. When a user opens a PowerPoint presentation containing hyperlinks using the Internet Explorer browser, they are not warned about possible malicious content by PowerPoint. Warnings to the user are dependent on the link's security mechanisms. For example, hyperlinks to files associated with executables (.exe files) cause a warning dialog to the user when activated. Hyperlinks to html files invoke the user's browser which then invokes its own security dialogs based on the browser's security configuration settings. Hyperlinks to other Office applications are dependent upon the security mechanisms in that application. For example, hyperlinks to Word documents containing macros are flagged to the user based on Word's security features.

2.4.2.5 ActiveX Controls/Objects

ActiveX controls, as well as other types of objects, including bitmap images, Word documents, or Excel spreadsheets, may also be inserted into a PowerPoint presentation (*Insert->Object)*. For example, the user can create an image that takes up an entire slide and set up a hyperlink to a Word document with embedded macros. If the user clicks on the image, the Word application fires and the embedded macros execute. The security mechanisms in this scenario are dependent upon the Word application.

The user may also choose to activate an object action by choosing the *Object action* radio button within the Action Settings dialog. The user is then offered a pull-down menu of choices. Available actions vary depending on the object, and the user may or may not be warned about

the execution. In one scenario, the user is warned for the *Activate* action as follows: *You are about to activate an OLE object that may contain viruses or be otherwise harmful to your computer*. The user is then given the choice to enable or disable the execution.

Customized ActiveX controls may also be added using the *Insert->Object* menu option and choosing the *Create from file* radio button. Although users are not warned that their presentation contains ActiveX controls when the presentation is opened, they do receive the warning, as shown above, about the dangers of activating OLE objects if they attempt to activate the embedded controls.

Standard ActiveX controls for the user interface are available from within PowerPoint and may be controlled by attaching VBA code. These controls may be added to the UserForm or directly to a PowerPoint document or master. (To add controls to the UserForm, refer to section 2.4.2.1. To add controls directly to the document, choose *View->Toolbars->Control Toolbox*.) The Control Toolbox offers controls similar to those available with the Controls offered within UserForms. Controls are added by dragging and dropping them to the document. Code may be attached to a control by double-clicking on the control and adding appropriate VBA statements to chosen events. If ActiveX controls are added using the Control Toolbox, the user is warned about macros when they open the containing presentation. In this case, the mere presence of the control is enough to activate the warning; actual macros may or may not exist.

Since the Office applications support the HTML format, references to ActiveX controls may also be added to a presentation which has been converted to HTML. The secure configuration of Internet Explorer is then essential as this browser is typically used to view HTML files. The High setting is recommended for all security zones, or customized settings that disable ActiveX.

ActiveX controls are especially dangerous, since they are separate executables with high capability. It is therefore very important to not activate ActiveX controls from untrusted sources. When VBA code is associated with objects that are inserted into a presentation, the user will see a warning dialog as shown in Figure 2.4.b.

2.4.2.6 Running Programs & Macros from Action Buttons

Action Buttons, as briefly described in section 2.4.2.4, are buttons that can be added to a presentation to provide some service. The service may be activated by using the Action Settings dialog as shown in Figure 2.4.d. The options include: *None, Hyperlink to, Run program, Run macro, Object action,* and *Play sound.* The *None* and *Play sound* options require no further discussion. The *Hyperlink to* setting was discussed in section 2.4.2.4 and the *Object action* setting was discussed in section 2.4.2.5. However, it is worthwhile to provide more discussion on the *Run program* and *Run macro* settings.

The designer may choose to run an external executable program in response to clicking or dragging the mouse over an Action Button by choosing the *Run program* radio button. For example, the Action Button may execute a Calculator program when activated. The security warnings to the user vary depending on the Service Releases installed on the machine. For example, PowerPoint 97 with no Service Releases **did not** issue a warning to the user about the dangers of running executable code. However, PowerPoint 97 with Service Release 2a

provided a warning dialog with the user option to enable or disable the macros when the user activated the Action Button.

VBA Macros may be constructed within the PowerPoint presentation (by choosing *Tools->Macro->Visual Basic Editor).* These macros may then be attached to an Action Button and run by choosing the *Run macro*: radio button. The user will not receive any type of warning when selecting the Action Button. However, when the PowerPoint presentation containing the macro is first opened, the user will see a warning dialog as shown in Figure 2.4.b. Once again, the user can choose to enable or disable the macros.

This capability with Action Buttons provides methods for running malicious executable programs and thus increases the threat potential. The good news is that the user will be confronted with a warning dialog on activation of Action Buttons linked to executable programs if the latest patches from Microsoft have been installed. Users should be aware of the security risks when running executables from Action Buttons without the latest releases installed as the user will not be warned of possible malicious code. In addition, macros attached to Action Buttons are flagged when the user opens the presentation as long as the macro checker remains enabled.

It should be noted that there is also at least one method for running an executable program that takes advantage of a vulnerability in Word. For example, a button's action may be set to run a Microsoft Word application. The Word application may contain embedded VBA macros which trigger upon the opening of the document. If the embedded macros are actually within the Word template for the document, then the user will not be warned about macros within the template unless the relevant patch from Microsoft is installed (SR-2 fixes this vulnerability). If the user wants to share this PowerPoint presentation with other users, perhaps by sending it in an e-mail message as an attachment, then the user can share not only the presentation and linked Word document, but also the executable code. By choosing to use the URL option under Hyperlinks, the user may choose to use the http protocol to reference the Word document.

2.4.2.7 Pack and Go Technology

PowerPoint comes with a *Pack and Go* wizard which is used to package PowerPoint presentations for other users (*File->Pack and Go...* menu option). If they don't have PowerPoint available, the package can include a PowerPoint Viewer. This technology was tested to see if macros could also be included in the packaged presentations and the security features included within the Viewer.

The Viewer is not capable of executing macros within PowerPoint presentations. However, the *Pack and Go* technology still is a security concern since the Viewer is an executable program (Ppview32.exe) that could be modified to include malicious instructions using reverse engineering techniques. Consequently, it is a security risk to accept *Pack and Go* presentations from untrusted sources.

2.4.3 Examples

Example 1: Simple PowerPoint VBA Macro Using Action Button

The example shown in Figure 2.4.e demonstrates a simple VBA macro written within the PowerPoint development environment. This example displays a message to the user when they click on the Action Button (shown to the left of the code) while in Slide view. Action Settings are used to call the macro named testMacro whenever the button is clicked by the user.

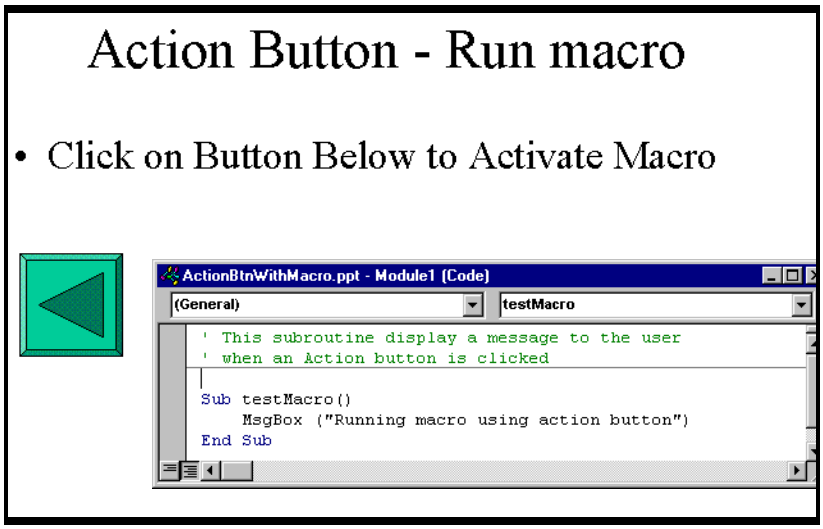


Figure 2.4.e: Example 1 - Macro attached to Action Button

Example 2: VBA Program using PowerPoint and Outlook Object Model

This example demonstrates several concepts, including macro activation mechanisms, use of the PowerPoint object model, and use of other Office object models such as Outlook. First, macros within PowerPoint may be initially executed using several methods as described in section 2.4.2. This particular example uses a toolbar button (not shown) attached to the macro called testOLE. When testOLE is activated, it executes its one instruction which shows the form pictured at the upper left of Figure 2.4.f. The form provides a GUI which includes a label and a listbox. Showing the form then triggers the UserForm_Initialize event which activates the code shown in the bottom portion of the diagram. This routine's sole purpose is to fill the listbox with shows currently playing in Las Vegas. If the user clicks on a show, then the ListBox1_Click macro is executed which uses Outlook to create and send an e-mail message informing the recipient of the show that was selected. The form is then hidden from the user. The user only sees the list of shows within the user interface. If the user never selects a show, then the macros associated with the click event are never executed. Otherwise, mail is sent without the user's knowledge.

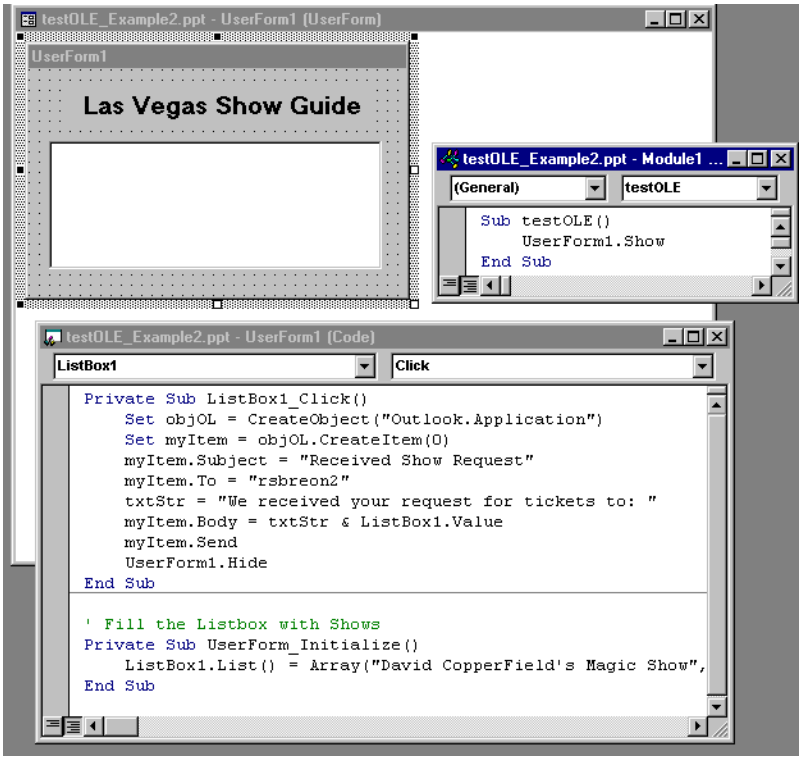


Figure 2.4.f: Example 2- VBA Macro using PowerPoint and Outlook Object Models

Example 3: Simple Attack Example

An example of a simple attack takes advantage of the Action Button vulnerability discussed in section 2.4.2.6. This vulnerability involves using Action Buttons to run executable programs within a PowerPoint presentation without any type of warning to the user. The vulnerability was discovered by the Internet community and has been patched by Microsoft. However, many users don't bother to install patches; consequently, there are probably many desktops running the vulnerable version of PowerPoint 97.

To exploit the vulnerability, a developer creates a new presentation or modifies an existing one. An Action Button may be added to the master or a document by choosing *Slide Show->Action Buttons*. The developer may then pick from the choice of icons presented and and draw the icon onto the document. In response to the Action Settings dialog, the developer may choose the *Run program:* radio box and enter their malicious executable in the textbox followed by selecting the OK button. The attacker should have this program execute both when the user clicks on the button as well as when the mouse is dragged over the button. The developer can improve covertness by enlarging the button to fill the slide master and changing the color to transparent. Text may then be added to the button or on the page behind the button to appear as if textual information is residing within a frame. Now the user will execute the program whenever they pass over the presentation slides with the mouse without even realizing that they are executing code, assuming the latest patches have not been installed.

2.4.4 Countermeasures

The macro checker is the fundamental security feature within PowerPoint that protects against executable content attacks. Unfortunately, the user may easily disable this feature through the user interface. It is imperative that the macro checker remain enabled. This can be verified by choosing *Tools->Options*; *General* tab; *Macro virus protection* checkbox.

Add-Ins may contain dangerous macros which execute without warning to the user. Unlike presentations or templates, these macros may trigger when the Add-In is opened. However, these Add-Ins must be loaded in order for them to execute. Add-Ins with embedded macros may be easily shared by e-mailing them as attachments to mail messages. When opened by the recipient, a macro warning from PowerPoint will fire if PowerPoint is installed and the macro checker is enabled. However, once the Add-In is loaded, the macros will execute whenever PowerPoint is opened without additional warnings to the user. The only countermeasure is to prevent initial loading of Add-Ins through macros by keeping the macro checker enabled and disabling macros from untrusted sources.

It is imperative that users install all security-related patches to secure their systems against known attacks. For example, programs may be run from Action settings without warning to the user if the latest patches to PowerPoint have not been installed on the user's system.

The PowerPoint Viewer may be packed with presentations and delivered to another user easily by using the *Pack and Go* Wizard that comes with PowerPoint. This Viewer is made available so that a user does not have to have the full PowerPoint application installed in order to run another person's presentation. However, the Viewer is an executable program which could be modified by an attacker to include malicious code. Consequently, users should be careful about accepting *Pack and Go* presentations from untrusted sources.

HTML support within PowerPoint offers threat potential from various types of executable content, including ActiveX controls. It is imperative that the Internet Explorer browser be securely configured to limit this threat potential. Typically, this means setting the security level to High for all security zones or customizing the settings to limit ActiveX and other executable content capability.

Third party products offer virus scanning capability. These products provide some protection against known viruses. However, macros within PowerPoint presentations are easily modified so that typical virus scanners will not catch them. Consequently, third party products offer a very limited solution at this time.

2.4.5 Summary of PowerPoint

PowerPoint 97 has attack potential due to its ability to include executable content in the form of VBA macros, ActiveX controls, hyperlinks, external executables, and other types of scripting available with the HTML format. It is important that the user implement the countermeasures as outlined above to help protect their systems against executable content attacks.

PowerPoint 2000 was released by Microsoft in June of 1999. This latest release requires investigation of new executable content vulnerabilities since it promises to be a widely-used product.

2.5 Outlook 98

2.5.1 Overview

Outlook is Microsoft's primary email client; however, it also offers other services such as calendaring and scheduling. Consequently, it is often described as a collaboration tool or groupware product since it facilitates the sharing of information among a group of people.

Although Outlook 97 is still packaged with Office 97, this section will concentrate on the more recent version, Outlook 98. Many of the executable content features in Outlook 98 are also available in Outlook 97 with the most major exception being HTML capability. In Outlook 98, the user may choose their mail format to be HTML, thus providing the capability of mailing web pages. This feature is not in Outlook 97. However, the other features which will be discussed, such as embedded macros within form events, are available in both versions. In addition, this research is based on the Outlook client configured with the Microsoft Exchange server, since this is the most popular configuration option.

Outlook includes the Visual Basic Scripting Edition (VBScript) version 3.0 programming language. VBScript offers a subset of VBA's capabilities to make it a "safer" language. For example, file access features have been stripped out of VBScript. VBScript is an interpreted language from Microsoft which relies on a host application, such as Outlook, in order to execute. It cannot run stand-alone. The core language remains the same regardless of the host application; however, various internal and external mechanisms may be available to the VBScript program depending on the host features as well as those of the environment. For example, Outlook 98 includes several methods and properties which may be called using the VBScript language. These methods and properties may be used to manipulate various Outlook items such as e-mail messages. In addition, VBScript may have access to other objects in the environment such as ActiveX controls or Object Linking and Embedding (OLE) controls. For example, VBScript may be used from within Outlook to manipulate Word documents on the user's platform using the Word Object model.

2.5.2 Threat Potential

The core VBScript language offers limited attack potential in that it offers no file or network access features or commands for launching a program. However, the Outlook host application and outside environment provide substantial attack capability. The Outlook host includes methods and properties to manipulate Outlook mail messages. Additional capabilities include the manipulation of the current user's environment. For example, VBScript can be used to activate ActiveX controls or manipulate other Microsoft objects available on the platform, such as Microsoft Word documents.

The possible types of compromises include using the Outlook object model to steal mail messages from a target or send messages on behalf of another user by masquerading as that user. The highly-publicized Melissa virus used a Word file attachment to call methods within the Outlook object model that accessed the user's Outlook address book. People listed in the address book were then programmatically mailed copies of the virus which many of these users then propagated on to people in their address book. This virus essentially shut down the mail system for several major companies.

Other types of possible compromises involve manipulating other objects available from within the user's environment. Embedded VBScript in an Outlook document may call methods from other Microsoft object models, such as other Office 97 applications or activate an ActiveX control. For example, VBScript may be embedded in an Outlook mail message that modifies the user's Word documents.

There is also more than one method for embedding script within Outlook mail messages. The method used impacts the security/countermeasures that are available. One method is to embed VBScript within form events. Outlook uses the concept of a form which contains the layout rules for the content. It also uses the concept of events, which are triggering mechanisms that correlate to user actions. If script is attached to an event, then that script executes when the associated trigger is activated. Outlook forms respond to a whole list of events including the opening, saving, and closing of a form. For example, a user may enter VBScript within the Item_Open event of a form. The user may then mail an Outlook email message and the embedded form to another user. When the recipient opens the mail message, the associated form will open thus executing the attached script. Since the script could be malicious, this is viewed as a security concern. Consequently, Microsoft added a security mechanism in the form of a dialog box that warns the user that the message contains embedded script which could be malicious.

Another method for including script within an Outlook mail message is to embed it within the HTML content of the message. This is a new feature in Outlook 98 which permits a user to send web pages to another user. To enable this feature, the user must set their mail format to HTML. Within the HTML, various types of scripting may be embedded, including VBScript, Javascript, and links to Java applets. Given the various programming languages that may be used to embed code within the HTML format of an Outlook mail message, the attacker has a wide range of capabilities at his disposal. The security mechanisms provided by Microsoft to prevent compromise from possible malicious script in this context includes High, Medium, Low, and Custom settings for different security zones. The High setting provides the most security while the Low (and sometimes Custom) security settings provide the least. The security mechanisms will be described in more detail in the Countermeasures section.

Outlook mail messages also can include file attachments. These attachments may include any type of executable content and are typically launched when the attachment is opened. The types of compromise are dependent on the users's Outlook security settings, the file extension used in the file attachment name, and the application used to open the file attachment. Microsoft provides a user-configurable security setting within its Attachment Security dialog. By default, this setting is High which means that the user is presented with a warning dialog for most file attachments. However, the user may also set the security setting to None which disables the warning mechanism. There are exceptions to this general security feature. File attachments associated with other Office 97 products, including Word, Excel, Access, and PowerPoint, do not trigger the attachment security mechanisms regardless of the setting. Also, the user may disable the security mechanism for certain file types through the Outlook 98 user interface so that any future user on that machine does not receive warnings for any file attachments with the given extension. Although this may be disabled using the interface, to re-enable it requires changing a value in the registry. However, the security mechanisms for file types that usually indicate executable code, such as files ending in .exe, cannot be disabled in

this manner. The application associated with the file extension of the attachment may provide additional security mechanisms. For example, Word provides its own warning dialogs to the user. If malicious executable content is embedded within a file attachment and the security mechanisms are thwarted, the code may perform many different types of activities, including the deletion, modification, or exfiltration of the user's data.

2.5.3 Examples

Example 1: Simple VBScript Macro in a Form

Example 1 demonstrates script within an Outlook form event called Item_Open. Script attached to the Item_Open event executes when the containing item, such as a mail message, is opened. This script presents a warning dialog box about the dangers of macros when the user opens the document using this form.

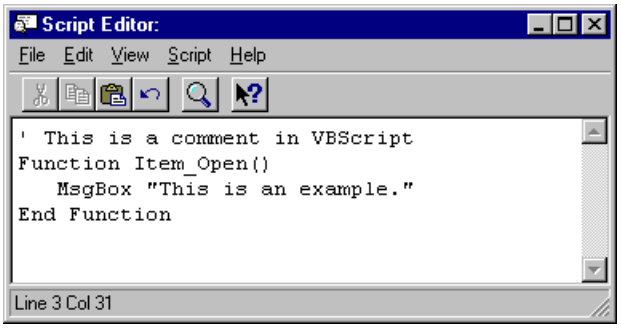


Figure 2.5.a: Example 1 - VBScript with Outlook Form Event

Example 2: VBScript within Form that uses Object Libraries

Example 2 demonstrates how to create a new mail message and send it to a user using VBScript code within a form. This code uses methods and properties from the Outlook Object library to create a new mail item, provide values for the Subject, To, and Body fields, and send the new mail message to the user specified in the To property. Once again, the script is embedded in an Outlook form using the Item_Open event as the triggering mechanism.

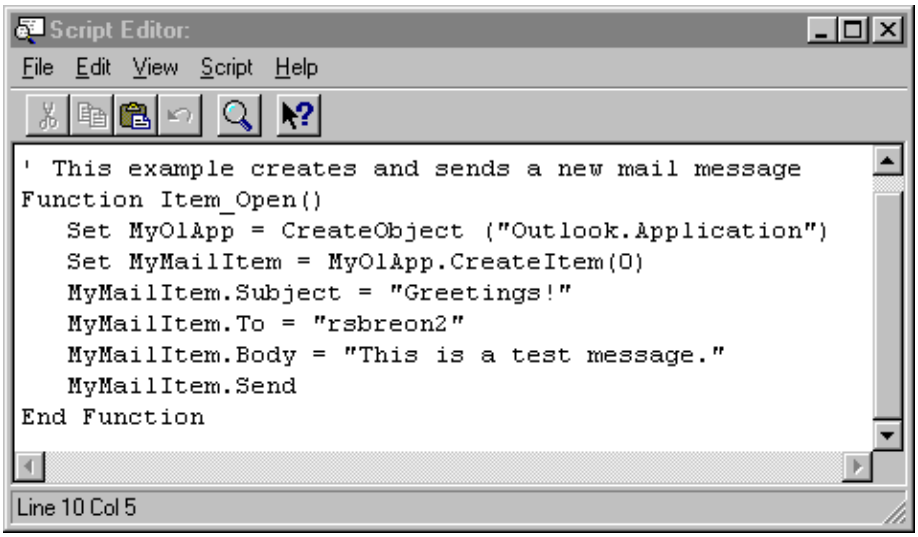


Figure 2.5.b: Example 2 - VBScript Using Outlook Classes Within Form Event

Example 3: VBScript Within HTML

Example 3 shows VBScript embedded within an HTML file. The file may then be inserted as part of the Outlook 98 mail message body (by using the *Insert->File*, selecting the HTML file from the file browser, and selecting the *Text only* option). This script is the same as that used in Example 2; however, the security mechanisms are different. This script will not invoke the warning mechanism used with forms. Rather, the security mechanism that may be triggered are based on the High, Medium, Low, or Custom settings associated with the various security zones found using either the Outlook Security interface (*Tools->Options; Security* tab) or Internet Explorer.

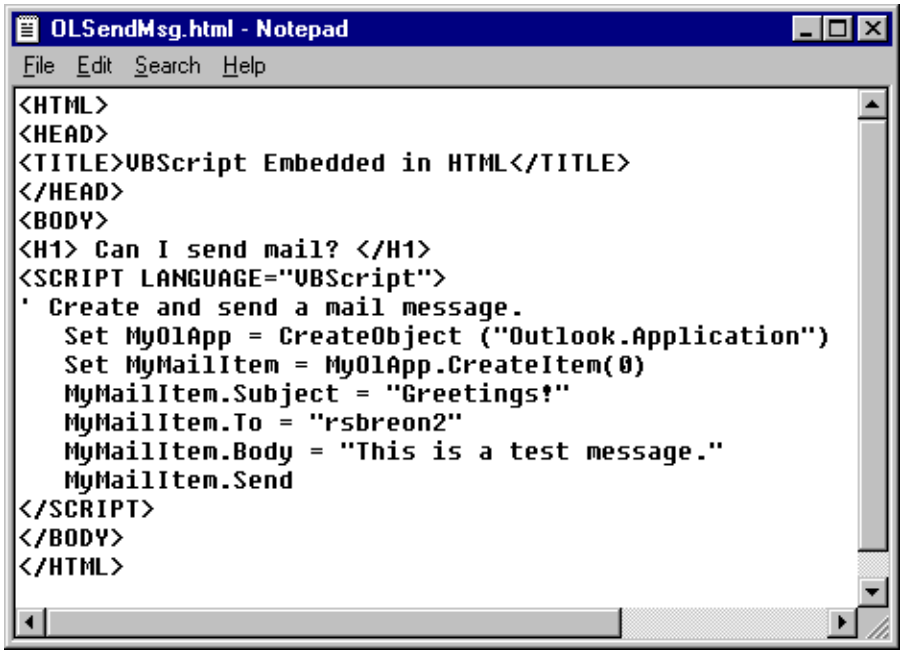


Figure 2.5.c: Example 3 - VBScript within HTML file

2.5.4 Countermeasures

As described briefly in the Outlook Threat Potential section, Outlook 98 has several security mechanisms to protect against the execution of embedded code within forms, web pages, and file attachments. Since executable content may perform malicious activities on behalf of the user, such as exfiltration, deletion, or modification of the user's data, it is important to use the security mechanisms provided and to be aware of the possible dangers.

To protect the user from possibly malicious script embedded in an Outlook form, the user is warned with a dialog box as shown in Figure 2.5.d below:

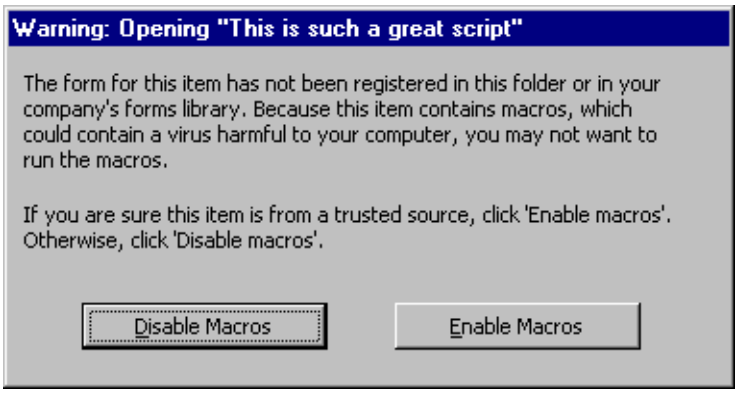


Figure 2.5.d: Dialog Warning Outlook Users of Macros Within Form

Unfortunately, Outlook does not distinguish between malicious and non-malicious content. Rather, the warning dialog is presented to the user whenever script is present. Consequently, users may get into the habit of consistently enabling the script. The security mechanism is not enabled if the script is embedded within a "trusted" form. A form becomes trusted when it is published to the Organizational Forms Library or a Public Folder. The permission to publish in these two areas must be granted by a Microsoft Exchange administrator. This is assuming the Outlook client is using the Microsoft Exchange server.

If script is embedded into a mailed web page using Outlook, the security is provided through the security zone settings (*Tools->Options* menu option/*Security* tab). Two security zones are available from the *Zones* pulldown menu: *Internet zone* (default) and *Restricted sites zone*. All e-mail will use the security settings associated with the zone selected. To achieve protection from active content, it is recommended that the user select *Restricted sites* for the e-mail zone and disable all options that provide active content such as ActiveX controls, Java applets, and scripting. (Actual recommended settings are shown in Appendix B.) While disabling active content for some users will prove too limiting, most users just require textual information within their e-mail messages. It should also be noted that these settings are the same as those in Internet Explorer and other Microsoft applications that implement security zones. In fact, changing the settings using the Outlook user interface will also change the settings in these other applications and vice versa.

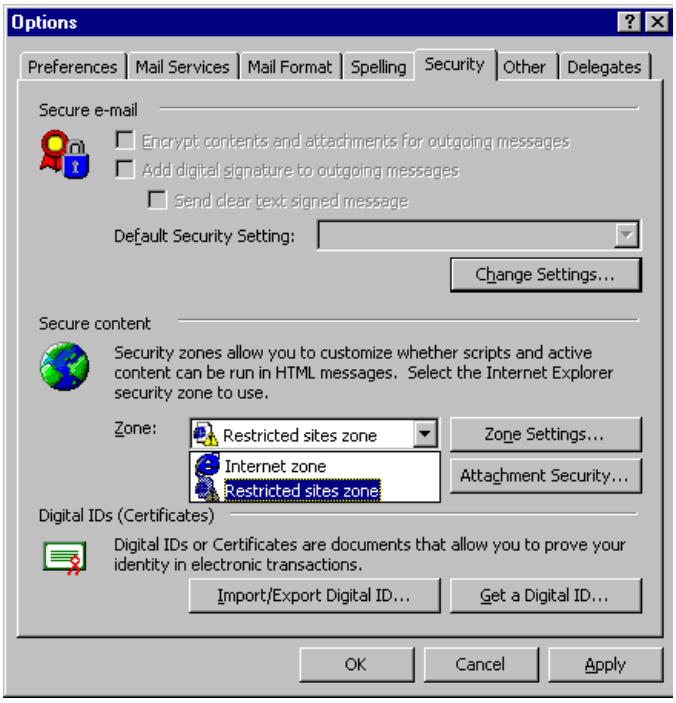


Figure 2.5.e: Security Zones for Outlook Mail

File attachment security is controlled through several mechanisms. The main mechanism is shown in Figure 2.5.f. The attachment security, by default, is set to High. With this setting, the user is warned about possible malicious content within file attachments for most file attachment types. However, as discussed in the Outlook Threat Potential section, there are exceptions. Documents created with other Office 97 products do not trigger the security dialog. Rather, Microsoft depends on the application used to open the file attachment to provide the security. Also, a user may disable the dialog for certain file types. Once disabled, it is disabled for the entire machine and it may only be re-enabled by resetting the proper value in the registry. The security dialog for file attachments that are flagged as executables by their file extension, such as .exe files, cannot be disabled through the user interface using this mechanism. However, if the Attachment Security is set to None, the user will not be warned about possibly malicious content regardless of the file type. It is therefore very important that the user retain the High Attachment security level and that this security not be disabled for any individual file type. It is equally important that the user be aware of the possible consequences associated with opening file attachments. File attachments may contain many and varied forms of malicious content which can wreak havoc on a user's system. Users should be aware of this threat and heed the warning dialogs; otherwise, the security mechanisms fail.

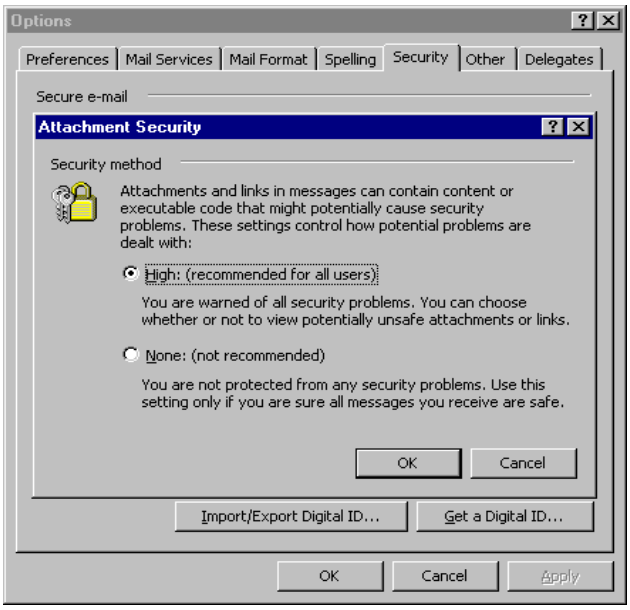


Figure 2.5.f: Attachment Security Dialog

2.5.5 Summary of Outlook

The Outlook client provides several vehicles for sending executable content to other users including forms, HTML web pages, and file attachments. The threat potential is high since Microsoft's object models support methods and properties for manipulating a user's Outlook mail as well as other objects available within the user's environment. In addition, the ability to embed content within the HTML of a web page expands the threat potential since the capabilities of other languages besides VBScript may be used, such as Java applets and Javascript. File attachments allow even more forms of executable content. To decrease the threat, it is important that the user maintain secure configuration settings. This includes using the Restricted sites security zone with active content disabled and maintaining the High Attachment Security setting as shown in Figure 2.5.f. In addition, the user should be extremely careful if choosing to enable script embedded in mail messages or in file attachments since even mail messages that appear to come from trusted sources may not be safe.

Future work should include a re-evaluation of Outlook 2000 as this product includes the more powerful VBA language, rather than VBScript, for customization.

3.0 Conclusions

The Office 97 applications pose considerable security risk to their users due to the customization capability within each application. This customization capability includes the option to

embed VBA/VBScript code, ActiveX controls, and hyperlinks to other locations. The following table summarizes the capabilities of each application.

Feature	Word	Excel	Access	PowerPoint	Outlook 98
VBA Programming Language	Yes	Yes	Yes	Yes	No
VBScript Programming Language	No	No	No	No	Yes
Macro Checker	Yes	Yes	No	Yes	Yes
Disable Macro Checker Using GUI	Yes	Yes	N/A	Yes	No
Enable Macro Checker Using GUI	Yes	Yes	N/A	Yes	No
Supports ActiveX Controls/OLE	Yes	Yes	Yes	Yes	Yes
Supports Hyperlinks to URLs	Yes	Yes	Yes	Yes	Yes
Supports HTML Format	Yes	Yes	Yes	Yes	Yes
Supports Auto Events	Yes	Yes	Yes	No*	No
Supports Templates	Yes	Yes	No	Yes	No
Supports Add-Ins	No	Yes	Yes	Yes	No
Supports Forms	Yes	Yes	Yes	Yes	Yes

* Although PowerPoint presentations do not support Auto events, PowerPoint Add-Ins do support them.

VBA, used within Word, Excel, Access, and PowerPoint, is an especially powerful programming language with instructions that can perform many different types of attacks, including:

- stealing another user's information by sending files back to an attacker as email messages.
- modification of a user's security settings by changing the registry.
- inserting and executing malicious programs.

Although VBScript's attack capability is much more limited, Outlook can use methods and properties available in the Office 97 Object Libraries to perform malicious activities such as the stealing of a user's email messages or sending email while masquerading as another user.

The main security feature implemented within the Office 97 applications is the macro checker. However, the macro checker can be easily disabled by the user through the User Interface provided within each application (except Outlook). Even when enabled (the default), users frequently ignore the warning dialogs about macros as demonstrated by the Melissa virus. Consequently, it is imperative that users are educated about the dangers of executable content so that the current security mechanisms are taken seriously.

Since each of the applications support ActiveX controls, hyperlinks to URLs, and HTML, it is also important to securely configure Internet Explorer. Typically, this translates to implementing the High setting for each security zone or customizing the settings such that ActiveX controls and other types of scripting are disabled.

It is also extremely important that users incorporate the latest security patches from Microsoft. Otherwise, the user's system is vulnerable to known attacks, frequently publicized over the Internet.

Commercial virus scanners also offer some protection against Office 97 macros. The main problem with most scanners is that they only protect against known viruses.

Future work should include a thorough analysis of Office 2000 as it promises to be a widely-used product by customers and hackers. It is especially important to research Outlook 2000 as this application now includes the more powerful VBA programming language rather than VBScript and offers an easy mechanism for deploying attacks to other users. In addition, commercial security products have entered the market which proclaim to identify certain types of executable content, such as ActiveX controls. These products require investigation to determine if they provide possible countermeasure solutions to the threat of mobile code.

4.0 Appendix A: Macros within a PowerPoint UserForm

Within PowerPoint, UserForms may be customized to include various user interface controls, such as buttons, text boxes, and checkboxes. Macros may be attached to either the form itself or the controls. The following gives an example of implementing a UserForm with controls and macros.

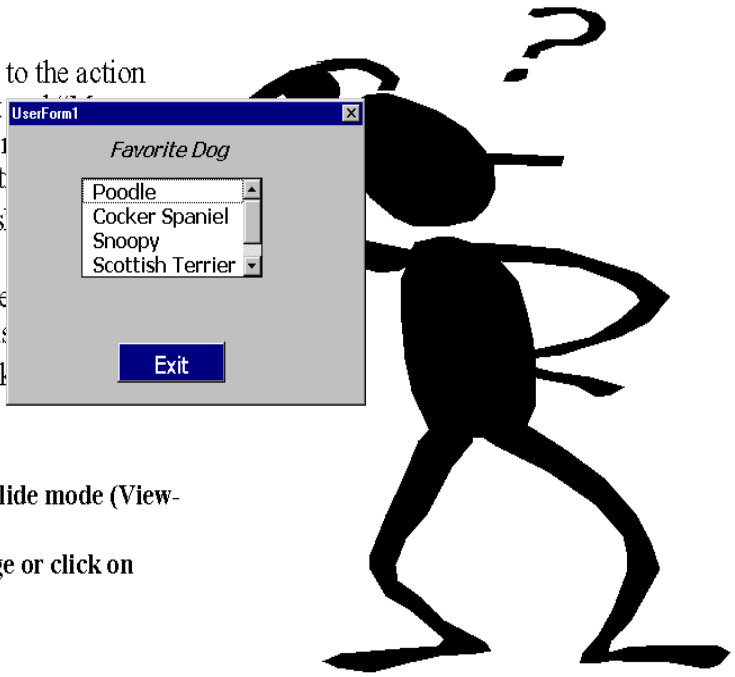
1. Open a PowerPoint presentation
2. Create a new slide (Insert->New Slide...)
3. Choose an AutoLayout for this slide. The example shown below used bullets at the left and an image at the right of the slide.
4. Create an image (Insert->Picture->From File...) - This example used Amconfus.wmf.
5. To create a new macro, choose Tools->Macro->Visual Basic Editor
6. Choose Insert->Module
7. Enter desired VBA code within the programming window. For example,

```
Sub testForm()
    MsgBox (“In testForm”)
    UserForm1.Show
End Sub
```

8. Then create a new UserForm by choosing Insert->UserForm
9. A UserForm called UserForm1 pops up along with a Controls toolbox. Controls may be added to the form by clicking on the desired control and then clicking on the form. The example uses label, command button, and listbox controls.
10. VBA Code may be added to these controls by double-clicking on the control. There is typically a series of possible events for each control.
11. Testing of this code may be done by choosing Run->Run Sub/UserForm
12. It is important to note that the UserForm will not be displayed to the user unless the Show property is used in the testForm subroutine.
13. Now attach Action Settings to the image in your PowerPoint presentation. This is done by selecting the image and choosing Slide Show->Action Settings...
14. Choose the “Run macro” radio box and insert the name of the macro that will start this chain of events called testForm. The user can set this macro to run when the user clicks on the macro and/or when the user drags the mouse over the image by choosing the appropriate tab.
15. Save the presentation by choosing File->Save
16. Macros within the PowerPoint presentation slide below will execute when the user clicks or drags the mouse over the image on the right. This causes the testForm macro to run since it is attached to this Action Setting. The testForm macro shows a message box and then shows the UserForm with the listbox and button controls.
17. To test the macros, run the slide show by choosing View->Slide Show

Using Action Settings, UserForm, and Controls

- Image at right is attached to the action settings of "Mouse Click
  Drag" which invokes the
  (Slide Show->Action Sett
- testForm is a macro that s
  UserForm
- Code attached to Initialize
  triggered when the form i
- Code attached to the Click
  ListBox component
- To start the macros:
  - **View presentation in slide mode (View->Slide Show)**
  - **Drag mouse over image or click on image**



5.0 Appendix B:Recommended Outlook Security Settings

The following is an excerpt from a technical report entitled, “Guide to the Secure Configuration and Administration of Microsoft Exchange”, written by Trent Pitsenbarger of C43. It provides recommendations for the secure configuration of the Outlook 98 client using Internet Explorer 5 (IE 5).

Security Zones

Outlook 98 can take advantage of Internet Explorer security zones to protect against malicious code (ActiveX controls, Java, or scripts) embedded into the body of messages. Internet Explorer includes a capability to restrict the execution of such code based upon four zones. Before jumping into how Outlook 98 uses these settings, a quick review of their use in Internet Explorer is in order.

Local Intranet zone: This zone contains addresses that are typically behind the organization’s firewall or proxy server. The default security level for the Local Intranet zone is “medium”.

Trusted Sites zone: This zone contains sites that are trusted -- sites that are believed not to contain files that could corrupt the computer or its data. The default security level for the Trusted Sites zone is “low”.

Restricted Sites zone: This zone contains sites that are not trusted -- that is, sites that may contain content that, if downloaded or ran, could damage the computer or its data. The default security level for the Restricted Sites zone is “high”.

Internet zone: By default, this zone contains anything that is not on the computer or an intranet, or assigned to any other zone. The default security level for the Internet zone is “medium”.

For each zone, one of four levels of restrictions can be enabled:

High: Do not execute
Medium: Warn before executing
Low: Run without warning
Custom: Establish custom settings

Outlook 98 utilizes these zones in that you can select which of two zones -- the Internet zone or the Restricted zone -- Outlook messages fall into. The settings for the selected zone are then applied by Outlook to all messages.

It is recommended to:

- Use the Restricted zone. To set the zone, select **Tools/Options** and the “Security” tab. Select “Restricted sites” from the zone drop-down box.
- Set the settings for the Restricted zone as recommended below by selecting “Zone Settings” and clicking on “Custom Level”. Note that changes made here will also apply to the Restricted zone when web surfing with Internet Explorer. These recommendations apply

specifically to Internet Explorer 5.0; the options available under Internet Explorer 4.0 are similar but do not include all of the settings.

- Download signed ActiveX controls - DISABLE
- Download unsigned ActiveX controls - DISABLE
- Initialize and script ActiveX controls not marked as safe - DISABLE
- Run ActiveX controls and plug-ins - DISABLE
- Script ActiveX controls marked safe for scripting - DISABLE
- Allow cookies that are stored on your computer - DISABLE
- Allow per-session cookies (not stored) - DISABLE
- File download - DISABLE
- Font download - DISABLE
- Java permissions - HIGH SAFETY
- Access data sources across domains - DISABLE
- Drag and drop or copy and paste files - PROMPT
- Installation of desktop items - DISABLE
- Launching programs within an IFRAME – DISABLE
- Navigate sub-frames across different domains - DISABLE
- Software channel permissions - HIGH SAFETY
- Submit nonencrypted form data - DISABLE
- User data persistence – DISABLE
- Active Scripting - DISABLE
- Allow paste operations via script - DISABLE
- Scripting of Java Applets - DISABLE
- Logon - Anonymous logon

Note that following these recommendations will disable many advanced features; however, for the vast majority of e-mail users there will be no operational impact. This is because most e-mail messages are simple text messages with attachments. The features that are disabled deal primarily with script and controls embedded within the body of the message which is not typically done.

Note once again that these settings are shared with the Internet Explorer browser and web pages typically DO incorporate the kinds of features which are disabled via these settings. While this could represent an operational impact, keep in mind that the Restricted zone is intended to include those sites that are not trusted - one should restrict what those sites can do and in fact these recommended settings are only slightly more restrictive than the default set-

tings for this zone. Also note that descriptions of these settings simply are not provided by Microsoft or documented in any known public documentation. As a consequence, we are investigating the settings further and may make some modifications to our recommendations as our efforts mature.

Using these settings will counter known attacks that use active content contained within the body of e-mail messages such as the BubbleBoy virus. Note that these settings do not guard against attacks contained within an e-mail attachment, such as Word macro viruses. File attachment security, which was discussed in the prior section, is applicable to this kind of threat.

6.0 References

[1] Brophy, Keith & Koets, Timothy, *Teach Yourself VBScript in 21 Days*, Sams.net Publishing, Indianapolis, IN, 1996.

[2] *Building Applications with Microsoft Outlook 98*, Microsoft Press, Redmond, Washington, 1998.

[3] Cassel, Paul, *Sams' Teach Yourself Access 97 in 14 Days,* Sams Publishing, 1996.

[4] *Executable Content: Definition, Taxonomy, and Examples*. C4 Executable Content Team, National Security Agency, 1996.

[5] Jaskolka, Karen and Gilbert, Mike, *Microsoft Office 97 Programming with VBA for Dummies,* IDG Books, 1997.

[6] Lomax, Paul, *VB & VBA in a Nutshell*, O'Reilly & Associates Inc., 1998.

[7] *Microsoft Office 97 Visual Basic Programmer's Guide*, Microsoft Press, Redmond, Washington, 1997.

[8] Mr. Bunny's Guide to ActiveX, Carlton Egremont III, Addison Wesley Longman, 1998.

[9] Network Associates webpage: http://www.nai.com.

[10] O'Brien, Timothy, *et al. Microsoft Access 97 Developer's Handbook,* Microsoft Press, 1997.

[11] Simard, Don and DeSiena, Tom, *The Visual Basic for Applications Executable Content Format*, C4 Executable Content Team, National Security Agency, 1997.

[12] Solomon, Christine, *Microsoft Office 97 Developer's Handbook*, Microsoft Press, Redmond, Washington, 1997.

[13] Stevenson, Nancy, et al., *Using Microsoft PowerPoint 97*, Que Corporation, Indianapolis, IN, 1997.

[14] VB & VBA in a Nutshell, Paul Lomax, O'Reilly & Associates Inc., 1998.

[15] Vermaat, Shelly Cashman, *Microsoft Office 97 Advanced Concepts and Techniques*, Course Technology, Cambridge, MA, 1998.

[16] Wells, Eric and Harshbarger, Steve, *Microsoft Excel 97 Developer's Handbook*, Micro Modeling Associates, Microsoft Press, Redmond, Washington, 1997.