Archived NIST Technical Series Publication

The attached publication has been archived (withdrawn), and is provided solely for historical purposes. It may have been superseded by another publication (indicated below).

Archived Publication

Series/Number:	NIST Special Publication 800-28
Title:	Guidelines on Active Content and Mobile Code
Publication Date(s):	October 2001
Withdrawal Date:	March 2008
Withdrawal Note:	SP 800-28 is superseded in its entirety by the publication of SP 800-28 Version 2 (March 2008).

Superseding Publication(s)

The attached publication has been **superseded by** the following publication(s):

Series/Number:	NIST Special Publication 800-28 Version 2
Title:	Guidelines on Active Content and Mobile Code
Author(s):	Wayne A. Jansen, Theodore Winograd, Karen Scarfone
Publication Date(s):	March 2008
URL/DOI:	http://dx.doi.org/10.6028/NIST.SP.800-28ver2

Additional Information (if applicable)

Contact:	Computer Security Division (Information Technology Lab)
Latest revision of the attached publication:	SP 800-28 Version 2 (as of June 19, 2015)
Related information:	http://csrc.nist.gov/
Withdrawal announcement (link):	N/A

Date updated: June 19, 2015

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

National Institute of
Standards and Technology
Technology Administration
U.S. Department of Commerce

Special Publication 800-28

Guidelines on Active Content and Mobile Code

Recommendations of the National Institute of Standards and Technology

Wayne A. Jansen

1901-2001
NIST CENTENNIAL

NIST Special Publication 800-28

Guidelines on Active Content and Mobile Code

Recommendations of the National Institute of Standards and Technology

Wayne A. Jansen

C O M P U T E R S E C U R I T Y

Computer Security Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20988-8930

October 2001

U.S. Department of Commerce
Donald L. Evans, Secretary

Technology Administration
Karen H. Brown, Acting Under Secretary of Commerce for Technology

National Institute of Standards and Technology
Karen H. Brown, Acting Director

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analysis to advance the development and productive use of information technology. ITL's responsibilities include the development of technical, physical, administrative, and management standards and guidelines for the cost-effective security and privacy of sensitive unclassified information in Federal computer systems. This Special Publication 800-series reports on ITL's research, guidance, and outreach efforts in computer security, and its collaborative activities with industry, government, and academic organizations.

National Institute of Standards and Technology Special Publication 800-28
Natl. Inst. Stand. Technol. Spec. Publ. 800-23, 53 pages (Oct. 2001)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 2001

For sale by the Superintendent of Documents, U.S. Government Printing Office
Internet: bookstore.gpo.gov — Phone: (202) 512-1800 — Fax: (202) 512-2250
Mail: Stop SSOP, Washington, DC 20402-0001

Table of Contents

Foreword v

Executive Summary v

Introduction 1

Background 1

- Browser Anatomy 3
- Server Anatomy 8

Threats 11

- Underlying Issues 14
- Categories of Threats 16

Technology Related Risks 17

- PostScript 19
- Portable Document Format 20
- Java 20
- JavaScript and VBScript 22
- ActiveX 22
- Desktop Application Macros 24
- Plug-ins 25
- CGI and Related Interfaces 26

Safeguards 27

- Security Policy 27
- Risk Analysis and Management 28
- Evaluated Information Technology 29
- Security Audit 30
- Application Settings 30
- Version Control 31

Incident Response Handling 31

Automated Filters 31

Behavioral Controls 32

Readers 32

Digital Signature 33

Isolation 33

Minimal Functionality 34

Least Privilege 34

Layered and Diverse Defenses 34

Summary 35

Terminology 37

On-line Resources 40

References 41

Annex A – HTTP Request Methods 46

Annex B – HTTP Response Status 46

Foreword

This document provides guidelines for Federal organizations' acquisition and use of security-related Information Technology (IT) products. These guidelines provide advice to agencies for sensitive (i.e., non-national security) unclassified systems. NIST's advice is given in the context of larger recommendations regarding computer systems security.

NIST developed this document in furtherance of its statutory responsibilities under the Computer Security Act of 1987 and the Information Technology Management Reform Active of 1996 (specifically section 15 of the United States Code (U.S.C.) 278 g-3(a)(5)). This is not a guideline within the meaning of 15 U.S.C. 278 g-3 (a)(3).

These guidelines are for use by Federal organizations that process sensitive information.[1] They are consistent with the requirements of OMB Circular A-130, Appendix III.

The guidelines herein are not mandatory and binding standards. This document may be used voluntarily by non-governmental organizations. It is not subject to copyright.

Nothing in this document should be taken to contradict standards and guidelines made mandatory and binding upon Federal agencies by the Secretary of Commerce under his statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, the Director of the Office of Management and Budget, or any other Federal official.

Executive Summary

The private and public sectors depend heavily upon IT systems to perform essential, mission-critical functions. As existing technology evolves and new technologies are introduced to provide improved capabilities and advanced features in systems, new technology-related vulnerabilities often arise. Organizations implementing and using advanced technologies, therefore, must be increasingly on guard. One such category of technologies is *active content*. Broadly speaking, active content refers to electronic documents that, unlike past character documents based on the American Standard Code for Information Interchange (ASCII), can carry out or trigger actions automatically without an individual directly or knowingly invoking the actions. Therefore, exploits based on vulnerabilities in active content technologies by their very nature can be

[1] The Computer Security Act provides a broad definition of the term "sensitive information," namely "any information, the loss, misuse, or unauthorized access to or modification of which could adversely affect the national interest or the conduct of federal programs, or the privacy to which individuals are entitled under section 552a of title 5, United States Code (the Privacy Act), but which has not been specifically authorized under criteria established by an Executive Order or an Act of Congress to be kept secret in the interest of national defense or foreign policy."

insidious. The following key guidelines are recommended to Federal departments and agencies for dealing with active content.

Federal departments and agencies should understand the concept of active content and how it affects the security of their systems.

The use of products, with capabilities for producing and handling active content, contributes to the functionality of a system as a whole and, thus, is an important factor in IT procurement and implementation decisions. Active content technologies allow code, in the form of a script, macro, or other kind of portable instruction representation, to execute when the document is rendered. Like any technology, active content can provide a useful capability for delivering essential government services, but it can also become a source of vulnerability for exploitation by an attacker.

Examples of active content include PostScript documents, Web pages containing Java applets and JavaScript instructions, proprietary desktop-application formatted files containing macros, spreadsheet formulas, or other interpretable content, and interpreted electronic mail formats having embedded code or bearing executable attachments. Electronic mail and Web pages accessed through the Internet provide efficient means for conveying active content, but they are not the only ones. Active content technologies span a broad range of products and services, and involve various computational environments including those of the desktop, workstation, server and gateway devices. Therefore, the knowledge required to understand their security ramifications is extensive. Federal agencies are encouraged to draw needed technical information from the many information resources that exist and gain sufficient understanding of the security implications of active content. Pointers to some useful on-line resources can be found in a separate section at the end of the document.

Federal departments and agencies should develop policy regarding active content.

Information security in any organization is largely dependent on the quality of the security policy and the processes that an organization imposes on itself. As appropriate to their situation, agencies should develop policy for the procurement and use of products involving active content technologies. A good criterion for decision-making is to apply active content where it specifically benefits the quality of the services delivered to the citizen and not simply for show or because of its availability within products. Both the consumption and production of active content should be addressed by the policy. A badly implemented, poorly planned, or nonexistent security policy on this subject can have a serious negative security impact, since over time these deficiencies have the potential to create a situation ripe for exploitation. The policy should be stated clearly and consistently, and made known and enforced throughout the organization. Putting an organizational security policy on active content in place is an important first step in applying effective safeguards and mitigating the risks involved.

Federal agencies should specifically be aware of the benefits they gain using active content and balance that against associated risks.

One of the most significant security practices often missing in an organization is the on-going process of risk analysis and management. Security involves continually analyzing and managing risks. A risk analysis identifies vulnerabilities and threats, anticipates

potential attacks, assesses their likelihood of success, and estimates the potential damage from successful attacks. Risk management is the process of assessing risk, taking steps to reduce risk to an acceptable level, and maintaining that level of risk. Experience has shown that use of active content technologies involves risk, since they are frequently accompanied by new vulnerabilities.

Security is relative to each organization and must take into account an organization's specific needs, budget, and culture. As new products are selected and procured, agencies need to consider the risk environment, cost-effectiveness, assurance level, and security functional specifications, in making their decisions. Agencies should also be aware of the interconnectivity and associated interdependence of organizations, and that a risk accepted by one organization may inadvertently expose other organizations, with whom they interoperate, to the same risk. Moreover, since active content is heavily oriented toward rendering information for an individual, their decisions may affect the citizens being served. Once an assessment is made, safeguards can be put in place against those risks deemed significantly high, by either reducing the likelihood of occurrence or minimizing the consequences of the attack.

Federal departments and agencies need to maintain consistent system-wide security when configuring and integrating products involving active content into their system environments.

Federal departments and agencies should be knowledgeable of the features in the products they procure, which can be used to control active content. Products and software applications that handle active content typically have built-in controls that can be used to control or prevent activation of related features. For example, The National Security Telecommunications and Information Systems Security Committee (NSTISSC) has recently issued an advisory memorandum on Web browser security[2] [NAM00] that outlines steps to lower associated risks through tightly controlled browser configurations. Electronic mail, spreadsheet, word processor, database, presentation graphics, and other desktop software applications have similar configuration settings that can be used to control the security capabilities of active content documents. Such configuration settings demand scrutiny in light of past exploits. Even today, many products are delivered with insecure default settings.

Network devices or other special purpose software should be used to supplement existing application-oriented controls. For example, firewalls can be augmented by gateway devices to filter certain types of electronic mail attachments and Web content that have known malicious code characteristics, and reject them at a point of entry. Desktop anti-virus software has also become increasingly capable of detecting malicious code signatures within active content. In addition, a new class of security product is emerging that dynamically restrains the behavior of mobile code by quarantining it within a logical sandbox. It behooves organizations to become familiar with all of the available security options and use them according to their organizational policy.

[2] http://csrc.nist.gov/publications/secpubs/index.html#other

Introduction

The private and public sectors depend heavily upon Information Technology (IT) systems to carry out essential, mission-critical functions. As existing technology evolves and new technologies are introduced to provide new capabilities and features, new vulnerabilities are often introduced as well. Organizations implementing and using advanced technologies, therefore, must be increasingly on guard.

One such category of emerging technologies is active content. Although the term has different connotations among individuals, it is used here in its broadest sense to refer to electronic documents that, unlike ASCII character documents of the past, can carry out or trigger actions automatically without the intervention of a user. Examples of active content documents include PostScript[3] documents; Web pages encoded in HTML or another markup language conveying or linking to mobile code such as JavaScript, VBScript, Java applets, or ActiveX controls; desktop application files containing macros; and HTML encoded electronic mail bearing executable attachments. Taken to its extreme, active content becomes, in effect, a delivery mechanism for mobile code. The purpose of this report is to provide an overview of active content, its technological underpinnings, and suitable security measures, so that the reader understands the associated security risks and can make an informed IT security decision on its application.

This report begins by providing background information on markup languages and other World Wide Web technologies involving active content. Readers already familiar with that material may wish to skip over the section. The discussion proceeds onto generic threats, followed by a perspective on risks drawn from past exploits involving technology-related vulnerabilities. Real-world examples appear throughout the report to increase understanding and awareness of the risks involved with various forms of active content. The report concludes by identifying available safeguards and summarizing some detailed recommendations. Key high-level recommendations appear at the front of the report in the executive summary. A glossary of relevant terms and links to useful on-line references appear at the end of this document.

Background

Being able to download files and electronic documents off the Internet is a useful function and a common practice for many people today. Web pages serve as an electronic counterpart to paper documents such as forms, brochures, magazines, and newspapers. Although paper documents come in different shapes and sizes, they are composed entirely of text and graphics. Similarly, most Web pages consist mainly of text and graphics. However, unlike paper documents, Web pages can entail active content, capable of

3 This document discusses certain computer manufacturers' products and standards. The discussion is not intended, however, to imply recommendation or endorsement, by the National Institute of Standards and Technology, nor is it intended to imply that the products and standards identified are necessarily the best available.

delivering digitally encoded multimedia information enlivened through embedded computer instructions.

Active content involves a host of new technologies such as built-in macro processing, scripting languages, and virtual machines, which blur the distinctions between program and data. Electronic documents have evolved to the point that they are themselves programs, or contain programs that can be self-triggered. Loading a document into a word processor can produce the same effect as executing a program and requires appropriate caution to be taken. The popularity of the World Wide Web (WWW) has spurred the trend toward active content. A dynamic weather map, a stock ticker, and live camera views or programmed broadcasts appearing on a Web page are common examples of use of this technology. Capabilities for Web access have also spread from workstations and desktop computers onto portable handheld devices, such as cell phones and Personal Digital Assistants (PDAs), and Internet appliances, affecting more people daily. Like any technology, active content can provide a useful capability, but can also become a source of vulnerability for an attacker to exploit.

Despite these capabilities, people tend to use electronic documents in much the same way that they use paper documents – accessing and viewing content, following references to other documents, and collecting information filled into forms. That is, Web pages delivered from Web servers to individuals via Web browsers impart an inherent document metaphor [Ven99]. The value of the document metaphor is that people are familiar with handling paper documents and can quickly adapt to using electronic facsimiles appearing within Web pages, since they understand the basic operations.

One drawback, however, is that the document metaphor is generally considered non-threatening and can lull one into a false sense of security. Moreover, strictly observing the document metaphor somewhat limits the way in which Web-based applications function. In particular, non-textual content does not lend itself to paper document style handling. For example, streaming or continuous delivery media, such as a live radio transmission or voice communication can transpire only as they occur in real time. In situations where the document metaphor has become awkward, alternatives that are more natural have arisen, evolving the document metaphor toward serving as a general-purpose vehicle to provision electronic services.

Code versus Data: In the recent past, the security risks associated with the use of computers were relatively straightforward. Instructions were distinct from the data on which they operated and some hardware could even distinguish internally between instructions and data (e.g., using a special memory bit), to avoid confounding them. Over the years the situation changed: tools emerged to facilitate application development in higher-level languages in lieu of machine languages, generic applications appeared, and hardware processing speeds increased dramatically. Eventually, it became advantageous to trade off the execution speed of compiled code against the flexibility of interpreted code.

An interpreter is a type of translator that accepts source code and executes it directly, without first producing object code (i.e., native machine instructions) as with a compiler. Reserved characters and/or words (e.g., "<" and “if” in JavaScript) are used by an interpreter to distinguish instructions within a text stream. An interpreter fetches each instruction sequentially, according to the flow of control, and carries out the intended behavior. Because of this, interpreters are inherently slower than

compilers, which directly generate machine instructions for execution. Interpretative languages range from lists of simple macro-type commands to complex programming structures. Special purpose interpreters, called emulators, have been used successfully to simulate the behavior of hardware no longer in existence or to provide a virtual machine environment for conceptual devices. The latter has been used effectively to support platform heterogeneity through interpretative compilers, which translate source code into virtual machine code that can execute on various independently implemented virtual machines.

With the arrival of the Web came the desire to make static pages more dynamic and lively by using interpreters throughout the system architecture. Today, most data files contain instructions that aid in the presentation or use of the data. Interpreters are ubiquitous: spreadsheet formulas, database query languages, word processing macros, and script interpreters not only embedded in Web browsers and servers, but also as stand alone development tools to forge applications from existing program components. While these technology improvements facilitate computer use, they also can involve serious risks, which are often not readily apparent to users. Many of these risks are associated with vulnerabilities created through the disguised (e.g., using special characters) or unexpected input of commands to an interpreter. For example, a program that uses an interpreter to process received information may be fed a string containing special characters that confuses the interpreter, causing it to treat correctly the initial part of the string as a legitimate parameter for some intended command, but treat incorrectly the remainder as a new unintended command sequence. Some applications use multiple interpreters in tandem, passing the output of one directly into another, which further compounds the problem, since a harmful operation may be brought about and executed unobtrusively along the way.

Browser Anatomy

A browser is the generic term used to refer to software that lets individuals view pages from various sources, including Web servers on the Internet, which make up the WWW. Netscape Navigator and Microsoft Internet Explorer are two popular Web browsers that aid in navigating text, graphics, hyperlinks, audio, video, and other multimedia information and services on the Web. Although Web browsers support a number of legacy protocols, such as the File Transfer Protocol [FTP], they rely mainly on a simple, request-response communications protocol, the HyperText Transfer Protocol (HTTP) [HTTP], to access Web servers. The browser requests information from a specific Web site, by sending a method request to the Web server conveying the Universal Resource Locator (URL) of the desired resource (e.g., a Web page), client information, and content handling capabilities. The URL is used to locate the server and serves as a unique address of the resource. Annex A contains a brief explanation of the available request methods a browser can issue. Typical usage involves mainly the issuing of GET and POST methods to retrieve information or provide form content, respectively.

Once the request is issued, the browser expects a response from the server, containing a status code, meta-information about the resource, the content corresponding to the resource requested (e.g., the Web page specified by the URL), and an indication of content encoding. Five general classes of response exist, as indicated by the first digit of the status code. For example, most users have received a 400 series code, the 404 code, at one time or another when unsuccessfully attempting to reach a resource at some site. Annex B contains a brief description of the classes of status code returned by a server.

In order to choose the best available representation of a resource at a browser, HTTP version 1.1 provides two forms of content negotiation:

- Server-driven, where the browser sends hints about its preferences to the server, using headers such as Accept-Language, Accept-Charset, etc., allowing the server to choose the representation that best matches the expressed preferences.

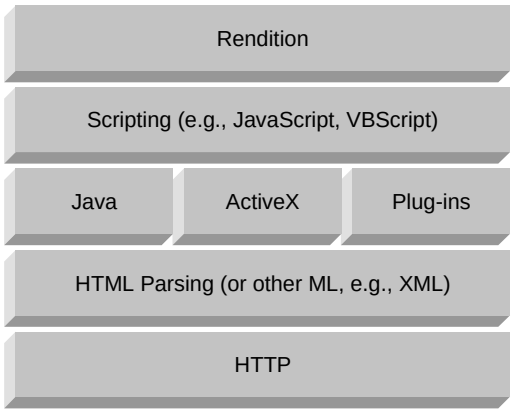
- Browser-driven, where in response to a browser request, the server replies with a list of the available representations and a description of their properties (e.g., language and character set), allowing the browser to choose one representation and reissue the request for the chosen variant.

The first alternative is the more mature. In fact, since the HTTP 1.1 specification does not completely define the headers needed for browser-driven negotiation, server-driven negotiation is the more viable alternative at this time.

The representation of a resource such as a Web page involves control codes, normally referred to as tags, and data. Browsers interpret control codes within Web pages, which denote the structure of the data (e.g., beginning of item, end of item) and the way to render it (e.g., heading, subheading, paragraph, list, embedded image). The codes may also embed URLs of additional information such as images, which entail further requests to the server to retrieve the information and complete the Web page. The control codes are the subject of intense standardization and include specifications for the HyperText Markup Language (HTML) [HTML4], Cascaded Style Sheets (CSS) [CSS1, CSS2], the eXtended Markup Language (XML) [XML1], and the Wireless Markup Language (WML) [WML]. Browsers are designed to read such codes, interpret their meaning, and render the Web page accordingly.

The original HTML specification, which signaled the birth of the WWW, also implicitly stipulates the basic requirements of a browser. Like most standards, commercial implementations have tended to extend the basic requirements into proprietary areas, and occasionally ignore a basic requirement or interpret it differently than originally intended. This has led to standardization bodies such as the WWW Consortium (W3C) to evolve standards along the lines of existing implementations, and developers of Web pages to undertake measures to ensure compatibility with versions of commonly used browsers.

Browsers inherently involve many different program components, both internal and external. Figure 1 illustrates the common components found in most browsers. The component layering illustrated is only for discussion purposes and not meant to imply any structural relationship. A basic protocol machine for HTTP, a parser for HTML, and a mechanism to render simple textual and graphical content are essential core components present in all browsers. The remaining components represent mechanisms to render other forms of content. To some extent, the specific choices depend on the browser manufacturer. However, competition and market demand influence manufacturers to offer components having comparable functionality with a high degree of compatibility and uniformity. Scripting languages (e.g., JavaScript, Visual Basic (VB) Script, and JScript), conveyed within the HTML from the server, are a useful means of having instructions executed by the browser, and require an interpreter for each language supported. Similarly, environmental components for Java, ActiveX, and Plug-in technology allow code external to the browser to be executed by the browser.



• Figure 1: Basic Components of a Generic Browser

To simplify the browser development, product designers allow extensibility through a variety of techniques for communicating with other functional components. The motivation is twofold: no one can reasonably build-in the means to render all forms of content themselves, and to attempt to do so would limit innovation as well as the usefulness of the browser. As long as the browser developer employs or provides a well-defined interface, other software manufacturers can readily extend functionality with their components. Overall, the program components of a browser, both built-in and otherwise, can be divided into the following classes [Mor98].

Program Components Incorporated Directly within the Browser

Browsers contain a significant amount of built-in functionality and typically can render a variety of content types inherently, including text, HTML delimited text, scripting languages, Java applets and common types of image files. The associated program components are functionally internal to the browser and able to interpret such content directly. To keep the browser safe from arbitrary content at sites that have diverse levels of trust, the program components must take adequate precautions against the information received. Because these programs are contained within the browser, the browser manufacturer is able to impose security constraints on them. Built-in functionality is also a means for the manufacturer to distinguish its product from others in various ways, such as offering proprietary extensions to standard script languages, close integration and interworking with other product offerings, and entirely new content handling capabilities.

In general, script-based languages do not incorporate an explicit security model in their design, and rely mainly on decisions taken during implementation. One noteworthy example of the rigor necessary is the implementation of secure JavaScript in Mozilla [Anu98]. The implementation controls access to resources and external interfaces, prevents residual information from being retained and accessible among different contexts operating simultaneously or sequentially, and allows policy, which partitions the name space for access control purposes, to be specified independently of mechanism. Java applets are also an interesting case, because the Java Virtual Machine (JVM), the internal browser component that provides the execution environment for Java applets, involves an elaborate level of security beyond that of the browser. The default security-policy settings for a JVM environment are normally determined by the browser manufacturer, but can be

tailored by each user. For example, with the Internet Explorer, a user can set Java permissions, such as file and network I/O, of both signed and unsigned applets received from various zones (i.e., trusted, intranet, internet, and restricted) to which specific Web sites can be allocated.

Program Components Installed to Extend the Browser via a Defined Interface

A significant innovation in the design of Web browsers is the ability to extend them beyond their built-in functionality. For a browser to hand off content rendering to such program components, the component must register its handling capabilities (i.e., the file extensions and MIME types it supports – see the sidebar below) with the browser when it installs. Often, these extensions require full access to the browser internals and the underlying operating system in order to accomplish their goals. Therefore, programs that extend browsers typically enjoy full-function interfaces to the internals of the browsers and to the operating system. The two most common means of extending browsers, Netscape plug-ins and ActiveX controls, have somewhat different security models. Microsoft ActiveX controls can require authenticated digital signatures as a prerequisite for installation, while Netscape plug-ins have no mechanism for enforcing authenticated signatures. However, once an ActiveX control is installed, it has free range over the entire machine, whereas the plug-in is typically confined to the capabilities of the browser.

MIME Types: Both browsers and Web servers are aware of Multipurpose Internet Mail Extensions (MIME) content types [MIME] and use them during content negotiation. MIME was designed originally as an extensible mechanism for electronic mail, using the convention of content-type/subtype pairs to specify the native representation or encoding of associated data fully. Content types include the following:

- Audio – used for transmitting audio or voice data.
- Application - used to transmit application data or binary data, and hence, among other uses, to implement an electronic mail file transfer service.
- Image – used for transmitting still image (picture) data.
- Message – used for encapsulating another mail message.
- Multipart - used to combine several body parts, possibly of differing types of data, into a single message.
- Text - used to represent textual information in a number of character sets and formatted text description languages in a standardized manner.
- Video – used for transmitting video or moving image data, possibly with audio as part of the composite video data format.

When a browser requests a Web page, it attaches information about what kind of content it can handle (e.g., "image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png") using MIME conventions. This allows the server to provide content selectively, based on the capability of the browser (e.g., serve graphics in GIF or JPEG instead of TIFF format).

In response to a browser's request, Web servers always indicate the type of content being sent via preliminary header information that conveys the MIME content type/subtype. Web servers use a mapping to associate the filename extensions of requested resources with MIME content types/subtypes. In a typical MIME map, each entry contains a filename extension to be associated with one or more MIME data types/subtypes. For example, an entry for image/gif uses the filename extension ".gif" and the entries for application/postscript use the filename extensions ".ps "and ".eps".

Based on the MIME types provided by the server, the browser must then ascertain what to do. It can render the associated content, either directly through a built-in or incorporated program component, or indirectly through the execution of a helper application. If the browser cannot locate a registered program entry for a MIME type, it can attempt to associate a program component using the filename extension of the resource and process accordingly. If that fails, as a final option, the browser can save the information to disk for later use. Note that for security reasons, browsers typically do not automatically launch executable programs downloaded from the Web, which have a MIME content type of "file/executable" indicated by the server. The default action is to ask whether the program should be launched or saved to disk, although the configuration can be set so that the browser starts such programs automatically without any prompts.

Programs Launched as an Independently Executing Process by the Browser

As an alternative to using a programming interface, a browser's capabilities can be extended using a so-called helper application or content viewer. Like a plug-in, the browser starts a helper application and hands off content rendering when it encounters a content type (MIME type or file extension) for which the helper application is registered to handle. Unlike a plug-in, a helper application runs separately from the browser in its own process space, and does not interact with or rely on the browser once initiated. Because a helper application runs independently, executed with the content file as input, it is completely outside the control of the browser, including the browser's security controls.

Program Components Directly Encapsulating the Browser

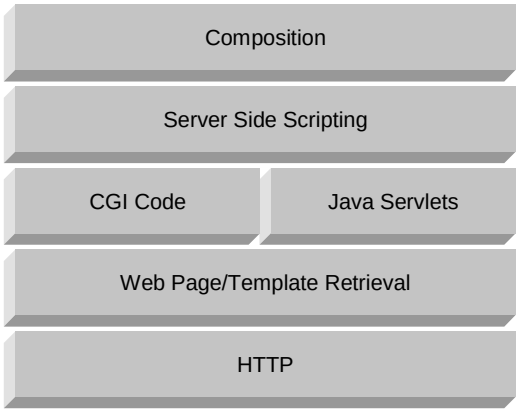
An interesting and somewhat unconventional approach is to embody the browser itself within another application to extend the functionality of the application. The best example is that Internet Explorer, or any other browser that complies with ActiveX container or OLE container technology, can be run as an ActiveX control inside an application. Visual Basic applications inherently have this capability, which allows one to not only control the URL requested, but also interact with the content received to the point of allowing HTML pages to pass information to and from the container application [Hug99]. Microsoft MSN, America Online, and a number of free Internet service access providers configure their browsers this way. A number of browsers, developed using different technology schemes, are available today specifically for being embedded within an application, as a way to add new functionality. An electronic mail application, for example, could use such a component to enhance its capabilities and render HTML formatted messages directly, including any embedded scripts.

Server Anatomy

A Web server is a program that resides on a computer on the Internet and supplies information and services formatted in HTML or another markup language, which contains text, image, audio, and video content. The primary function of a Web server is to respond to requests sent to it from a Web browser via HTTP.

The transaction begins when a browser requests a resource from the Web server. In the simplest case, the Web server retrieves the requested content from a file system and transmits it to the browser. While this approach works well for static non-volatile information, it can be unsuitable in situations where the information is volatile, already resides in a database or other repository under a different format, or varies according to the input provided. In such cases, the Web server responds to the request by creating the content dynamically, typically by spawning a process or lightweight thread to generate the information. The Common Gateway Interface (CGI) is an industry standard for communicating between a Web server and another program, often employed in such instances.

As with browsers, Web servers involve many different kinds of program components and are designed to be extensible and to interact with databases, legacy systems, and other servers running on an organization's network. Figure 2 illustrates the common components found in most Web servers, depicted using the same convention as was done earlier for browsers. The key components present in all Web servers are a basic protocol machine for HTTP, a means to fetch Web pages or resource templates, and a mechanism to compose and validate the contents of the response. The remaining components represent mechanisms to extend functionality, mainly for the purpose of generating information dynamically. Besides the widely supported CGI standard, the specific choices depend on the Web server manufacturer.



• Figure 2: Basic Components of a Generic Web Server

A CGI application executes as a separate process, which can be written in a variety of programming languages. As an independent process, the application is capable of accessing other hosts (e.g., a database server) and resources in performing its function, subject to its system security permissions. Once the application creates the information, the Web server conveys it in a response back to the browser. One drawback with this

approach is that it consumes a significant amount of computational resources to spawn a new process for each request. The kernel receives an interrupt each time the application is called; the application must then be allocated memory, loaded into memory, passed the input parameters from the browser, and executed. If the application is called multiple times in unison from different sources, multiple copies of the application are resident in memory simultaneously.

Because of the overhead involved with CGI, developers have sought after more efficient means to communicate between the software on the Web server and another program or library. A number of other programming interfaces offering performance improvements have arisen, such as the Netscape Server Application Programming Interface (NSAPI) and the Microsoft Internet Server Application Programming Interface (ISAPI). An application built using one of these interfaces, however, operates quite a bit differently from a CGI application. For example, rather than executing as an external application in a separate process, an ISAPI application executes as an integral part of the Web server within the same address space as the server code. This is possible because, unlike an external program, the ISAPI application is a dynamic link library component, which can be loaded or unloaded at will. Thus, the ISAPI application can remain in memory indefinitely or, when it finishes, be unloaded from memory and reloaded again, if needed, to conserve system resources. An ISAPI application is also a shared multithreaded code image, requiring only a single copy to support multiple simultaneous browser requests. These characteristics conserve system resources and improve response.

The techniques for dynamically generating content and improving Web server capabilities tend to be proprietary and used by software manufacturers to differentiate their product from others in the marketplace. Often the approach is to adapt or extend a technology developed for the browser environment, such as a particular scripting language or a framework for distributed program components, for the server environment. The following items provide examples of a few of the more notable technologies.

Server Side Includes (SSI)

SSI is a limited server-side scripting language supported by most Web servers. SSI provides a set of dynamic features, such as including the current time or the last modification date of the HTML file, as an alternative to using a CGI program to perform the function. When the browser requests a document with a special file type, such as ".shtml", it triggers the server to treat the document as a template, reading and parsing the entire document before sending the results back to the client. SSI commands are embedded within HTML comments (e.g., <!--#include file="standard.html" -->). As the server reads the template file, it searches for HTML comments containing embedded SSI commands. When it finds one, the server replaces that part of the original HTML text with the output of the command. For example, the SSI command given above (i.e., #include file) replaces the entire SSI comment with the contents of another HTML file. This allows the display of a corporate logo or other static information prepared in another file to occur in a uniform way across all corporate Web pages. A subset of the directives available allows the server to execute arbitrary system commands and CGI scripts, which may produce unwanted side effects.

Microsoft Active Server Pages (ASP)

ASP is a server-side scripting technology from Microsoft similar to SSI, which can be used to create dynamic and interactive Web applications. An ASP page is essentially an HTML template that contains server-side scripts that run when a browser requests an ".asp" resource from the Web server. The Web server processes the requested page and executes any script commands encountered, before sending the composed result to the user's browser. Both JScript and VBScript are supported scripting languages, but other scripting languages can be accommodated as well, provided an ASP compliant interpreter for that language is installed. For example, scripting engines are available for PERL, REXX, and Python languages from various sources. Scripting capabilities can be extended through the use of ActiveX objects, which can be developed in a variety of languages, including Visual Basic, C++, Cobol, and Java. A script that invokes an ActiveX object causes the object to be created and supplied any needed input parameters. Note that ActiveX is an optional technology not required by Active Server Pages.

Java Servlets

Servlets are based on Java technology and are essentially a kind of server-side applet. The Web server first determines whether the browser's request requires dynamically generated information from a servlet. If so, the Web server can then locate or instantiate a servlet object corresponding to the request (e.g., by uploading the code from another server) and invoke it to obtain the needed results. The Web server typically populates itself with the servlet objects, which remain active until invoked. Thus, no startup overhead is associated with execution of the servlet objects. A Web server may also offload the handling of servlets to another server. By relying on Java portability and observing a common applications program interface, servlet objects can run in nearly any server environment. Servlets support an object-oriented environment on the Web server, which is flexible and extendible. Moreover, untrusted servlet objects can be executed in a secure area, with the dynamically generated information being passed from the secure area into the remaining server environment.

Web Scripting – Client vs. Server: To understand Web scripting, one must distinguish scripts run by the browser (i.e., client-side scripting) from those run by the Web server (i.e., server-side scripting). Client-side scripting and server-side scripting are distinct concepts that serve different purposes. For example, since a server does not interact directly with a user, server-side scripting requires no human-to-computer interface capability. Furthermore, the Web browser and server each supply their own unique environment for executing scripts.

Client-side scripting is used to make Web pages more interactive and functional after they have been sent to the browser. For example, client-side scripts might involve validation of data entry fields on an HTML form so the user gets immediate feedback when a mistake occurs, or integration of an ActiveX control or Java applet with another component on the page so that they interact.

A Web browser environment for client-side scripting includes the objects that represent the user interface (e.g., windows, menus, dialog boxes, text areas, anchors, frames, cookies, and input/output) and a means to associate scripting code with events at that interface (e.g., change of focus, selection, loading and unloading of text and images, form submission, error and abort, and mouse

actions). Scripting code appears within the HTML and the displayed page is a combination of fixed and computed text, images, and user interface elements. Since the scripts react to user interaction, a main program is not needed.

A Web server provides a different environment for scripting, which includes objects representing requests, clients, and files, and mechanisms to lock and share data. All server-side scripting takes place before the resource (e.g., a Web page) is sent to the browser. The server-side scripts, for example, may involve creating a Web page dynamically by querying a database and formatting the results into HTML for delivery to the browser.

By employing both client-side and server-side scripting for a Web-based application, needed computations can be split appropriately between the browser and Web server environments, while providing a customized user interface.

Client-side scripting depends on the capabilities of the browser that processes a script, which requires awareness of what type of browsers might be encountered. While server-side scripting, such as with ASP pages, can create pure HTML pages acceptable by any browser, they are not necessarily portable to or compatible with the Web server software running elsewhere in an organization.

Threats

The openness of the Internet makes it easily accessible to intruders. Over recent years, intruder activity has revealed a number of shortcomings in the original design of the Internet. While some security features were foreseen and built into the relevant protocols, including the keystone Internet Protocol (IP), others were not addressed. They include the following omissions:

- *Data confidentiality:* Data passed across the Internet travel in packets that can easily be captured and viewed to reveal their contents.
- *Data integrity:* Data traversing the Internet may be intercepted and modified before reaching the recipient or replayed later.
- *User identification and authentication:* The responsibility for user authentication falls to the connected hosts. Unfortunately, many systems still rely on cleartext passwords and are, therefore, open to having them captured by an eavesdropper.
- *System identification and authentication:* Internet addresses identify host systems connected to the Internet, but because they are not authenticated or strongly bound to a host, the addresses can easily be spoofed.
- *Reliable domain name translation:* The Domain Name System (DNS) used to translate names to host addresses on the Internet, relies on truthful and accurate reporting of mappings by all components, which is difficult to ensure.

Intruders have used these omissions to attack hosts on the Internet. Furthermore, the Internet amplifies risks associated with vulnerabilities in connected hosts, by exposing those vulnerabilities to a broader range of threats.

An attack is a realization of some specific threat that affects the confidentiality, integrity, accountability, or availability of computational resources. Many initiatives to mitigate these threats are underway or reaching maturity. Standards for Internet Protocol Security (IPSec), Secure DNS, and Public Key Infrastructure (PKI) have been completed and realized in products. Commercial, government-certified, security evaluation laboratories have been established under regional and worldwide mutual recognition schemes. Competition for an algorithm to replace the aging Data Encryption Algorithm is complete, with NIST's selection of the Rijndael algorithm for the Advanced Encryption Standard. Organizations have established emergency response teams, which have improved their effectiveness in combating intrusions. Commercial software for detecting and eliminating computer viruses, filtering network protocols (i.e., firewalls), and detecting intrusions is widely available.

While the outlook should be positive, a number of factors contribute to perpetuating security problems.

Scale of the Internet

Millions of computers and tens of thousands of individual computer networks make up the Internet, precluding wholesale upgrades to new security protocols and solutions. Therefore, a large pool of systems lags behind in various degrees from having the most update protection mechanisms in place. While one's enterprise may have up-to-date and secure systems, one or more systems from the pool of stragglers may be used as a launch pad for new attacks, such as hard-to-prevent, distributed denial-of-service attacks.

Rate of Exposure

Invariably errors of commission or omission occur that allow protection mechanisms to be bypassed or disabled, and create a vulnerability. A vulnerability in and of itself may or may not pose a serious problem, depending on what tools are available to exploit it. As vulnerabilities are discovered and made known to the manufacturer, a window of exposure exists until a patch, containing corrective code to close the vulnerability, can be made available. A patch usually involves the installation of native, platform-specific code modules, which are developed as a replacement for or an insertion into compiled code already installed. Any delay in applying the patch opens the window not only to a wider time period for exploitation, but also to a greater audience of potential intruders, as attack tools that exploit the vulnerability emerge. Today, vulnerabilities are discovered at an increasingly high frequency, further compounding the problem.

Quality of Software

Tolerating the prevalence of large numbers of unintentional implementation errors in software products has become an accepted business practice. Modern market-driven development processes, such as synchronize and stabilize [Cus99] and extreme programming [Bec99], evolved to meet the demand for constructing large, complex feature-rich software products in a flexible manner. While such approaches emphasize efficient adaptability to incorporating new technologies, shifting priorities, and competition-driven features, these benefits come at the expense of discipline (e.g., formal design, code review, and complete testing) and schedule. The goal of producing a shippable product takes precedence over the elimination of known errors. While most errors are benign with respect to security, an unresolved implementation error may create a

serious security vulnerability. The lack of quality control manufacturers have over the implementation process is also indicated by the significant and growing occurrance of hidden functionality, so-called Easter Eggs, that exist in well known and widely used commercial products [Wer99]. Easter Eggs often embody significant code ranging from simple games to three-dimensional flight simulators. While intended to be non-threatening and surreptitiously honor the software development team whose names are eventually revealed, they provide another possible avenue for attack.

Confounding of Programs and Data

In striving to offer greater functionality and flexibility, software developers continue to blur the distinctions between program and data. While the intentions of the developer are presumably good, they can often have a negative impact when the need for security is not considered fully. Moreover, the prevalence of unintentional implementation errors in software applications that process electronic documents plagues active content technology. Even if a design is correct and secure, the implementation may unintentionally contain a serious vulnerability that can be exploited by malicious code conveyed in an active content document. An attacker needs only to learn what software their target is using, find an appropriate exploit, and send the document to the target.

Complexity of Software

The trend in application software development is to add more features and greater complexity to products. Greater complexity requires more code and more interaction among components, resulting in more implementation errors occurring. This trend combined with the competitive pressures facing manufacturers to be first to the market, the technical and cost barriers to extensive testing, and a marketplace that chooses functionality over security, ensures attackers continual opportunities in the future.

Configuration of Software

For a general population often baffled by the programming interface of a videocassette recorder, understanding a system's security posture and correctly setting its configuration is an unrealistic expectation. Yet, we are increasingly relied upon to exercise such skills, particularly with our own Web browsers and desktop software configuration. Even knowledgeable, enterprise system administrators are faced with a similar challenge – confronted with an array of security solutions, including those involving company proprietary and incompatible mechanisms, they must oversee a fragile patchwork of software products and devices that demand constant oversight.

Privacy Practices of Industry

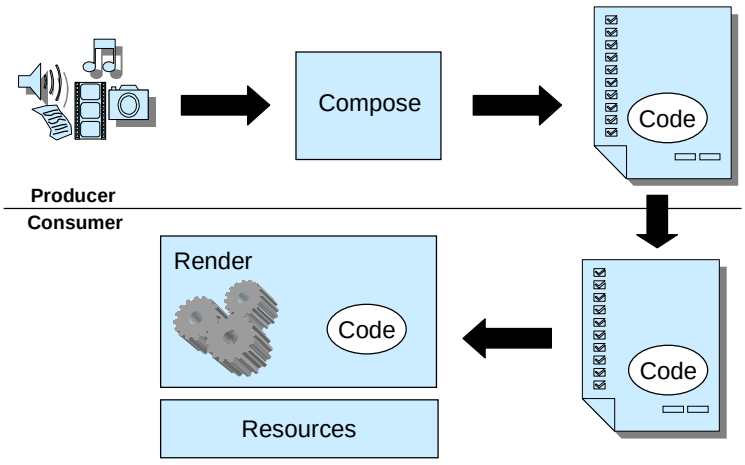
Commercial companies are increasingly using their Internet offerings to collect information on individuals, both directly (e.g., during credit card purchases and free service subscriptions) or indirectly (e.g., via persistent cookies, Web bugs, and spyware). These actions compound the aforementioned security problems, since successful attacks launched against commercial servers can also seriously affect the privacy of individuals whose information resides there.

Limits of Safeguards

Defending against all attacks completely in an open network such as the Internet is not possible. At the very least, the possibility exists for the occurrence of remotely launched denial-of-service attacks, which consume resources and deny the processing of legitimate requests by flooding the target with bogus requests. Recent distributed denial-of-service attacks, launched simultaneously from multiple proxy sites under the control of a single attacker, demonstrate both the ease in which the capacity of any Web site can be overwhelmed, and the value of having complete site redundancy for critical services.

Underlying Issues

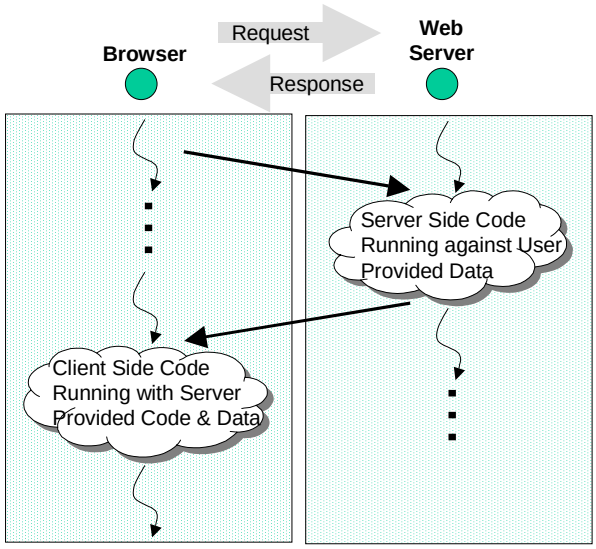
Active content allows code, in the guise of a script, macro, or other kind of portable instruction representation, to execute when the document is rendered. HTML and other related markup language documents, whether delivered via the WWW or another means, provide the richest mechanisms for conveying executable content; they are ideal active content containers. However, it is important not to lose sight of the fact that many other document formats, while not as rich in mechanisms, have similar potential. A more general view of the production and handling of active content is the producer-consumer model illustrated in Figure 3. Here, a producer composes an electronic document containing some form of executable code in addition to text, graphic images, audio, and video content. A producer could range from a user creating macros for a word processing document to a Web server dynamically generating Web pages. By some means, the document becomes available to a consumer who renders it with an electronic device within an execution environment, such as that provided by a word processor or Web browser. The execution environment of nearly all active content technologies impose policy restrictions that limit the code's access to computational resources during a rendition of the document.



• Figure 3: Producer-Consumer Model

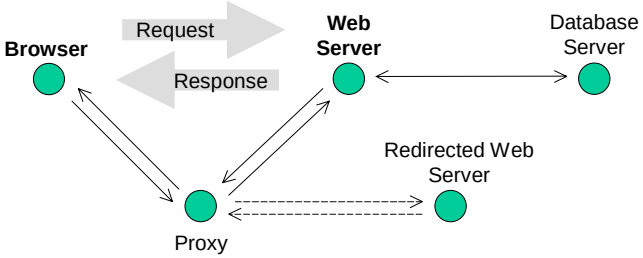
While any means used to compose, deliver, and render active content automatically is a concern, our focus is on the WWW, because the associated technologies are designed and implemented to work together seamlessly under this framework, with a user often unaware of the security implications. Figure 4 illustrates a simplified Web-based transaction supported by HTTP's client-server architecture. HTTP enables active content to be conveyed from one platform (i.e., a Web server) to another (i.e., a client browser)

where it is rendered automatically. As depicted in the figure, not only can the server impact the browser by supplying it code to execute, but also the reverse, input from the browser can influence the execution at the server. In the previous sections, we have seen that the process of rendering the active content is more involved than what is illustrated. Nevertheless, conceptually the depiction is accurate, insofar as the supplier of the code and the operator of the execution environment can be from different domains, posing a security risk for each side.



• Figure 4: Simplified HTTP Transaction

Web-based transactions often involve other computer platforms besides those of the browser and Web server. As mentioned earlier, the Web server may rely on a database server or other on-line repository or computational engine to fulfill requests it receives. HTTP also accommodates the use of a proxy, which normally resides on a firewall device to screen browser interactions, and automatic redirection by the referenced Web server to another server, which supposedly has the requested resource. Therefore, though we conceptually visualize a transaction as a simple HTTP interchange between a browser and a Web server, the reality is typically more complex, involving other entities. Figure 5 gives a more thorough picture of some of the entities that could be involved in an HTTP transaction.



• Figure 5: Entities Involved in HTTP Transaction Processing

Fred Cohen [Coh95] summarized some of the fundamental security issues of the design of the WWW, which bear repeating here:

- *Distributed untrusted computation:* "As a basic premise, the Web provides a means for information provided by arbitrary servers at unknown locations operated by unknown organizations to be interpreted by any of a large number of different browsers at unknown locations operated by unknown organizations. The idea of interpreting unknown information from unknown sources seems inherently risky."

- *Remote execution of untrusted software:* "Many Web extensions are designed to provide added function making the Web more than just a massive uncontrolled distributed database. These extensions, such as PostScript, Java, and MIME essentially allow for remote execution of untrusted software. For the browser, the risk is that the computer running the browser will be taken over, while for a server, the same risk extends to the server and any subsequent browsers that get information from that server once it is attacked."

- *Remote interpretation of unstructured and unverified content:* "In essence, most browsers and servers assume that the incoming information follows the HTTP protocol, but there is inadequate enforcement of this by servers and browsers. The result is that any incoming information might not conform, might be interpreted using an undefined method (corresponding to a don't care condition in the interpreter), and might result in arbitrary undesirable side effects."

Categories of Threats

A number of generic security threats apply to systems on the Internet, for example, unauthorized release of information, modification of information, and denial of service. The WWW, as a superstratum over the basic Internet technology, is subject to these same threats. In addition, the capabilities for supporting active content and mobile code provide new threat opportunities that fall within these general categories. Like any technology, active content can provide a useful capability, but can also become a source of vulnerability for an attacker to exploit. Overall, three different classes of attacks against this framework exist. While our emphasis is on the first of these, browser-oriented attacks, the others are important since a successful attack there may provide an avenue for launching a browser-oriented attack. Keep in mind too that other desktop applications, particularly electronic mail clients, often possess browser-like capabilities that are susceptible to browser-oriented attacks.

Browser Oriented

Attacks can be launched against Web browser components and technologies by the Web server. The mobile code paradigm requires a browser to accept and execute code developed elsewhere. Incoming code has two main lines of attack. The first is to gain unauthorized access to computational resources residing at the browser (e.g., security options) or its underlying platform (e.g., system registry); the second is to use its authorized access based on the user's identity in an unexpected and disruptive fashion (e.g., invade privacy or deny service). Because browsers can support multiple associations with different Web servers as separate windowed contexts, the mobile code of one context can also target another context. Unauthorized access may occur simply through a lack of adequate access control mechanisms or weak identification and authentication controls, which allow untrusted code to act or masquerade as a trusted component. Once access is gained, information residing at the platform can be disclosed or altered. Besides sensitive data, this information could include the instruction codes or configuration of the platform.

Depending on the level of access, complete control of the platform may be subsumed by the mobile code. Even without gaining unauthorized access to resources, malicious code can deny platform services to other processes by exhausting computational resources, if resource constraints are not established or not set tightly. Otherwise, the code can interfere with the platform by issuing meaningless service requests wherever possible.

Server Oriented

Attacks can be launched against Web server components and technologies by the browser. A browser can easily isolate and capture a response from a server, and may launch an attack by manipulating information and feeding back unexpected input to the server in a subsequent request. The idea is to induce the server into performing unauthorized commands provided by the browser, which in turn gains access to sensitive information or control of the server. For example, because HTTP is stateless, having no efficient means of maintaining persistent information between transactions, Web-based applications often use tricks, such as hidden fields within a form, to provide continuity between transactions, which may provide an avenue of attack. Similarly, other user provided input might eventually be passed to an application interface that interprets the input as part of a command upon which other commands can piggyback or whose interface buffer can be overrun in a buffer overflow attack. Such exploits may involve the complete analysis and reversing engineering of transactions by an attacker. Subtle changes introduced into the Web server can radically change the server's behavior (e.g., turning a trusted entity into malicious one), the accuracy of the computation (e.g., changing computational algorithms to yield incorrect results), or the confidentiality of the information (e.g., disclosing collected information).

Network Oriented

Attacks can be launched against the network infrastructure used to communicate between the browser and server. Even assuming the browser and Web server are well behaved, other entities may attempt actions to disrupt, harm, or subvert the framework. An attacker can gain information by masquerading as a Web server using a man-in-the-middle attack, whereby requests and responses are conveyed via the imposter as a watchful intermediary. Such a so-called Web spoofing attack [Fel97] allows the impostor to shadow not only a single targeted server, but also every subsequent server accessed. Other obvious attack methods lie outside the browser-server framework and involve targeting either the communications or the supporting platforms. For example, at a level of protocol below HTTP, an entity may eavesdrop on messages in transit between a browser and server to glean information. An attacking entity may also intercept messages in transit and modify their contents, substitute other contents, or simply replay the transmission dialogue later in an attempt to disrupt the synchronization or integrity of the information. Denial-of-service attacks through available network interfaces are another possibility, as are exploits involving any existing platform vulnerability.

Technology Related Risks

Risk is a measure of the likelihood and the consequence of events or acts that could cause a system compromise, including the unauthorized disclosure, destruction, removal, modification, or interruption of system assets. Most computer technologies involve some

degree of vulnerability due mainly to flaws or weaknesses in their design or implementation, or to the way in which they are deployed. More generally, a security vulnerability is the absence of or weakness in the security controls for a system that could result in a violation of the system's security policy. While technology-related vulnerabilities are often subtle and do not affect either the overall functionality or performance of a product, they can be discovered and exploited by an attacker. The impact of a vulnerability to an individual or organization is the subject of a risk analysis, and can vary widely, depending on such factors as the value of the resource affected or the perceived harm to one's reputation.

Two main factors should be considered when attempting to gauge the risk involved with active content technologies: the capabilities or facilities of the programming language, and the breadth and strength of controls in place within the execution environment to enforce policy [DoD00]. Different technologies offer a wide range of functionality regarding the types and granularity of actions on local resources such as windows, files, and network ports. Similarly, security controls can be imposed at different places within the execution environment and at different times (i.e., before or during execution), resulting in varying degrees of effectiveness. For instance, JavaScript, implemented within a browser environment, normally denies the ability of a downloaded script to read or write files by making file objects unavailable. In contrast, an ActiveX control, once accepted, can have complete reign over the entire file system. Besides the two main factors, determining the acceptable level of risk may involve other factors, such as the maturity of the technology, the scope of its intended use, the experience of the organization with the technology, and other controls in place.

To understand the range of risks associated with active content better, some popular active content technologies and their associated vulnerabilities are described below. They are provided merely as examples and do not imply an official opinion by NIST of the product or underlying technology. Most of the technologies discussed provide a useful capability when used in a Web environment. However, they also can be exploited by an attacker. The motivation for these technologies is to improve functionality and gain flexibility for the user. In a Web application, this often involves moving code processing away from the Web server onto the client's Web browser. Allowing remote systems to run arbitrary code on some local system, however, poses serious security risks. Traditional client-server systems do not involve such risks since they rely on static code on both the server and client sides.

Privacy Risks: Privacy is not synonymous with security; privacy is related to security, but is quite a different property. One may securely transmit personal or credit card information to a company, but who has access to the information after receipt is generally unknown to the individual. Although privacy breeches directly affect individuals, they can also affect the companies for which affected individuals work. For example, the inkling, unsubstantiated or not, that a company's CEO is suffering from a serious illness can cause its stock value to plummet.

Some organizations link records from different sources to target marketing efforts and to assess risks. When taken collectively, such information constitutes an electronic dossier on an individual, which in the wrong hands can cause harm, even if it is not completely accurate. No one can learn the full extent of the information kept on them by various organizations, much less verify accuracy, or control access. Sadly, much of the information collected over the Internet occurs in the background,

without the individual's knowledge or consent. One compelling example is the recent discovery that many legitimate companies have distributed free plug-ins or other software products containing so-called spyware – functionality that, once installed, periodically sends reports back to the company about its use and its environment.

Besides spyware, Web site developers have a myriad of tools at their disposal to collect personal information. They including tracking use by storing persistent information known as cookies on a system via a user's browser; embedding invisible single-pixel images within HTML, so-called Web bugs, whose downloading signals viewing to a third party; and invoking the communication capabilities of embedded scripts and program components downloaded.

Servers routinely log information that identifies users indirectly by recording client host names and even, when available, user names, and gives information about the request. Users may not be aware that such logs are being collected and most likely have no idea how that information is used or how long it is retained. There are few legal rules or ethical guidelines in most countries governing the disposition of log information, such as the sale to other organizations where they may be combined with other databases (e.g., on-line address listings) to infer further information.

PostScript

One of the earliest examples of active content is PostScript document representation [Ado99], still in wide use today. PostScript is a page description language from Adobe that is a de facto standard in commercial typesetting and printing houses. PostScript commands are language statements in ASCII text that are translated into the printer's machine language by a PostScript interpreter built into the printer. PostScript can also be interpreted by software on most computer platforms and drawing to computer screens or an attached drawing device. The interpreter uses scalable fonts, eliminating the need to store a variety of font sizes.

A PostScript file contains a document description, which is specified in the PostScript page description language. The language is a powerful interpreted language, comparable to many programming languages. Thus, PostScript documents inherently entail active content. For example, the language defines primitives for file manipulation, which can be used in a PostScript document to modify arbitrary files when the document is displayed or printed. Unfortunately, the operations can be abused by intentionally embedding malicious file commands within an otherwise harmless image, so that in displaying the image the interpreter also causes damage.

An early exploit of PostScript technology involved the language's ability to set a password held by the interpreter. In some hardware implementations of the language interpreter, if the password were set, it remained in non-volatile memory and prevented subsequent documents from being printed unless they contained the same password. An attacker sending a password-setting document could disable the printer in this way, requiring hardware replacement to rectify the situation [Cle90, Spe90]. Some PostScript interpreters can be set to disable potentially harmful primitives. For example, ghostscript, a well-known PostScript interpreter, recognizes the command-line option "-dSAFER" that disables file operations as well as the PostScript %pipe operator, which could be abused to

cause damage. One drawback is that applying such safeguards can also inhibit useful functions. This dilemma is a recurring theme with active content.

Portable Document Format

Portable Document Format (PDF) [Ado00a] is a page description language from Adobe for specifying the appearance of pages containing text, graphics, and images, using the same high-level, device-independent imaging model employed by PostScript. Unlike PostScript, however, PDF is not a full-scale programming language and does not include language features such as procedures, variables, and control constructs. PDF readers, used to view or print PDF files, can be installed as either a plug-in or a helper application for a browser. Other, full-featured tools exist to generate and manipulate, as well as view and print, PDF files.

A PDF document can be regarded as a hierarchy of objects. For example, a page object, which includes references to the page's contents (i.e., a content stream), other attributes, such as its thumbnail image, and any associated annotations, represents each page of the document. A content stream, in turn, is an object whose data consists of a sequence of instructions that describe the graphical elements to be rendered on a page, which are also represented as PDF objects using the same object syntax as the rest of the PDF document. However, whereas the document as a whole is a static, random-access data structure, the objects in the content stream are intended to be interpreted and acted upon sequentially.

Because of the object orientation and limited image-rendering operators, PDF is generally considered a benign format for use with a capability-limited content reader or viewer, such as the widely used Adobe Acrobat Reader. However, full-featured PDF tools, such as Adobe Acrobat, may be more susceptible to attack by virtue of their extended functionality. Recently an individual dramatically demonstrated that while the format itself may be benign, a PDF file could bear malicious code as an embedded file attachment [Fis01]. When the contaminated PDF file is opened, a game is launched that prompts the user to click on a moving image of a peach. The occurrence of that event, in turn, causes the execution of an embedded VBScript file, which attempts to mail out the PDF file to others using Microsoft Outlook.

Note that even content readers are not completely immune from problems. From time-to-time, vulnerabilities have occurred in the implementation of Acrobat Reader that could be exploited with carefully constructed content [Hir99]. For example, a recently released patch to Acrobat Reader 4.05 eliminates a buffer overflow vulnerability [Ado00b]. An attacker could exploit the vulnerability by creating a PDF file that would cause a Windows version of the reader to crash or to execute arbitrary code when the file was rendered. This example illustrates how even relatively benign content can affect document rendering software having implementation errors.

Java

Java is a full-featured, object-oriented programming language compiled into platform-independent byte code executed by an interpreter called the Java Virtual Machine (JVM). The resulting byte code can be executed where compiled or transferred to another Java-enabled platform (e.g., conveyed via an HTML Web page as an applet). Java is useful for adding functionality to Web sites. Many services offered by various popular Web sites

require the user to have a Java-enabled browser. When the Web browser sees references to Java code, it loads the code and then processes it using the built-in JVM.

The developers of Java tried to address the problem of security and were largely successful. The Java programming language and runtime environment [Gon98, Gos96] enforces security primarily through strong type safety, by which a program can perform certain operations only on certain kinds of objects. Java follows a so-called sandbox security model, used to isolate memory and method access, and maintain mutually exclusive execution domains. Java code such as a Web applet is confined to a sandbox, designed to prevent it from performing unauthorized operations, such as inspecting or changing files on a client file system and using network connections to circumvent file protections or people's expectations of privacy.

Security is enforced through a variety of mechanisms. Static type checking in the form of byte code verification is used to check the safety of downloaded code. Some dynamic checking is also performed during runtime. A distinct name space is maintained for untrusted downloaded code and linking of references between modules in different name spaces is restricted to public methods. A security manager mediates all accesses to system resources, serving in effect as a reference monitor. Permissions are assigned primarily based on the source of the code (where it came from) and the author of the code (who developed it), which restricts the access of the code to computational resources. In addition, Java inherently supports code mobility, dynamic code downloading, digitally signed code, remote method invocation, object serialization, and platform heterogeneity. Limitations of Java to account for memory, CPU, and network resources consumed by individual threads [Cza95] and to support thread mobility [Fug98] have been noted.

Hostile applets still pose security threats even while executing within the sandbox. A hostile applet can consume or exploit system resources inappropriately, or cause a user to perform an undesired or unwanted action. Examples of hostile applets exploits include denial of service, mail forging, invasion of privacy (e.g., exporting of identity, electronic mail address, and platform information) and installing backdoors to the system. The Java security model is rather complex and can be difficult for a user to understand and manage, which can increase risk. Moreover, many implementation bugs have also been found, which allow one to bypass security mechanisms [SUN01].

C# - An Emerging Technology: C# (pronounced C sharp) is a strongly typed, object-oriented programming language that enables programmers to build applications for the new Microsoft .NET platform. Under the .NET framework, languages compile into a common intermediate language, Microsoft Intermediate Language Code (MSIL), and use a common type and object system. MSIL enables the runtime environment, called the Common Language Runtime (CLR), to provide common services such as cross-language integration, cross-language exception handling, and security. C# borrows most of its operators, keywords, and statements from C++ and is intended to be used in programming both hosted and embedded systems. C# is normally compiled into MSIL byte codes and then just-in-time compiled into native code during execution, but can also can be compiled directly into native code. The language and security features appear to be similar to Java's and, thus, should pose similar risks. While Java runs on any platform having a JVM, C# currently runs only on Windows platforms.

JavaScript and VBScript

JavaScript is a general purpose, cross-platform scripting language, whose code can be embedded within standard Web pages to create interactive documents. While JavaScript is implemented widely today in Web browsers, it is also implemented within other execution environments on the client side, such as Acrobat Forms and Windows Script Host, as well as on the server side. Each context supplies the needed objects to control the execution environment and, therefore, the functionality can differ significantly from one context to another. Within the context of the browser, the language is extremely powerful, allowing prepared scripts to perform essentially the same actions as those that a user could take. Within that context, however, JavaScript does not have methods for directly accessing a client file system or for directly opening connections to other computers besides the host that provided the content source. Moreover, the browser normally confines a script's execution to the page with which it was downloaded.

The name JavaScript is a misnomer since the language has little relationship to Java technology and rose independently from it. For example, while JavaScript is an object-oriented language, it is based on prototypes, not on classes as with Java. Netscape developed JavaScript for its Navigator browser, and eventually JScript, a variation of JavaScript, appeared in Microsoft's Internet Explorer. Standardization, regulating the core language and facilities of JavaScript and JScript, resulted in the ECMAScript Language Specification [ECMA99], which both companies support in their products. Design and implementation bugs have been discovered in the commercial scripting products of both manufacturers.

Visual Basic Script (VBScript) is a programming language developed by Microsoft for creating scripts that can be embedded in Web pages for viewing with the Internet Explorer browser. Netscape Navigator, however, does not support VBScript. Like JavaScript, VBScript is an interpreted language able to process client-side or server-side scripts. VBScript is a subset of the widely used Microsoft Visual Basic programming language and works with Microsoft ActiveX controls. The language is similar to JavaScript and poses similar risks.

In theory, confining a scripting language to boundaries of a Web browser should provide a relatively secure environment. In practice, this has not been the case. Many browser-based attacks stem from the use of a scripting language in combination with a security vulnerability. The main sources of problems have been twofold: the prevalence of implementation flaws in the execution environment and the close binding of the browser to related functionality such as an electronic mail client. Past exploits include sending a user's URL history list to a remote site, and using the mail address of the user to forge electronic mail. The increasing use of HTML and other markup languages as content for electronic mail and in push technology services has opened new avenues for exploits through embedded scripts.

ActiveX

ActiveX is a set of technologies from Microsoft that provide tools for linking desktop applications to the World Wide Web. ActiveX controls are reusable component program objects that can be attached to electronic mail or downloaded from a Web site. ActiveX controls also come preinstalled on Windows platforms. Web pages invoke ActiveX controls using a scripting language or with an HTML OBJECT tag. It is possible to

specify a URL where the control can be obtained, if not installed locally. Unlike Java, which is a platform-independent programming language, ActiveX controls are distributed as executable binaries, and must be separately compiled for each target machine and operating system.

ActiveX Technologies: ActiveX is an ambiguous term, since it refers to a set of technologies under a common banner. Web users normally encounter ActiveX technology in the form of ActiveX controls, ActiveX documents, or ActiveX scripting.

- ActiveX controls, formerly known as Object Linking and Embedding (OLE) controls, are components (or objects) of prepackaged functionality that can be inserted into a Web page or other application for reuse. ActiveX controls are included with Microsoft Internet Explorer to allow Web pages to be enlivened with sophisticated formatting features, special effects, and animation.

- ActiveX documents allow an ActiveX-enabled Web browser to open an application, with the application's own toolbars and menus available, and serve as its container. This allows non-HTML native-formatted files, such as Microsoft Excel or Microsoft Word files, to be opened and manipulated seamlessly when encountered by the browser.

- ActiveX scripting refers to enhancements to VBScript and JavaScript to interact with ActiveX controls. ActiveX scripting can be used to integrate the behavior of several ActiveX controls and/or Java applications from the Web browser or server, extending their functionality.

The ActiveX security model is considerably different from the Java sandbox model [Ste00]. The Java model restricts the permissions of applets to a set of safe actions. ActiveX, on the other hand, places no restrictions on what a control can do. Instead, ActiveX controls are digitally signed by their author under a technology scheme called Authenticode. The digital signatures are verified using identity certificates issued by a trusted certificate authority to an ActiveX software publisher. For an ActiveX publisher's certificate to be granted, the software publisher must pledge that no harmful code will be knowingly distributed under this scheme. The Authenticode process ensures that ActiveX controls cannot be distributed anonymously and that tampering with the controls can be detected. This certification process, however, does not ensure that a control will be well behaved. Thus, the ActiveX security model assigns the responsibility for the computer system's security to the user.

Before the browser downloads an unsigned ActiveX control, or a control whose corresponding publisher's certificate was issued by an unknown certifying authority, the browser presents a dialog box warning the user that this action may not be safe. The user can choose to abort the transfer, or may continue the transfer if they assume the source is trustworthy or they are willing to assume the risk. Users may not be aware of the security implications of their decision, which may have serious repercussions. Even when the user is well informed, attackers may trick the user into approving the transfer. In the past attackers have exploited implementation flaws to cover the user dialogue window with another that displays an unobtrusive message such as "Do you want to continue?" while exposing the positive indication button needed to launch active content. More recently, an individual revealed another avenue for attack – the Authenticode certificates themselves –

by successfully posing as an employee of a well-known software manufacturer and receiving a class 3, software publisher certificate from a respected certification authority for public key certificates [Ver01]. This exploit opened the door for possible production and distribution of malicious ActiveX controls, appearing to come from the manufacturer.

An ActiveX-enabled browser provides an ideal vehicle for malicious code delivery. Once downloaded, the control is automatically executed without the victim having to take any conscious action. Members of the Chaos Computer Club, an infamous German hacking group, developed and demonstrated an ActiveX control that, under the pretense of displaying a graphic image to the user, accessed a popular accounting software package installed on the user's computer and transferred money from the user's bank account to their own [CNET97, Gil97]. Recent versions of Internet Explorer allow the user to customize the behavior of ActiveX controls depending on whether they are downloaded from a site on the Internet, a site on the local area network, or a site belonging to sets of identified trusted and untrusted sites.

Desktop Application Macros

Developers of popular spreadsheet, word processing, and other desktop applications created macros to allow users to automate and customize repetitive tasks. A macro, in its simplest form, is a series of menu selections, keystrokes, and commands recorded and assigned a name or key combination. When the macro name is called or the macro key combination is pressed, the steps in the macro are executed from beginning to end. Macros are used to shorten long menu sequences as well as to create miniature programs within an application. More complex macro languages often include programming controls (IF, THEN GOTO, WHILE, etc.) and language features that make them comparable to a scripting language. A virus can be written into a macro stored in a spreadsheet or a word processing document. When the document is opened for viewing or use, the macro is executed and the virus is activated. It can also attach itself to subsequent documents that are saved with the same macro. For these reasons, under normal circumstances desktop applications should not be configured to open automatically for another desktop application, such as a browser or electronic mail client, which receives untrusted content.

The recent Melissa virus is an example of the risk involved [Sha99]. A Microsoft Word document containing a malicious Visual Basic for Applications (VBA) macro propagated itself through the Internet by sending the host document as an electronic mail attachment addressed to contacts found in the previous victim's address book. A contact opening the attachment permits the macro to execute and the virus to take hold. VBA is an integral part of MS Office applications, included as a means for developers to build custom solutions within that environment. VBA is a superset of VBScript and offers the same automation and customization capabilities, but within the context of a desktop application. At least one other vendor, Corel, licenses and supports Microsoft's VBA in its products.

The newer generation of electronic mail applications, including the ones built into Web browsers, support HTML content and MIME attachments. Since active content provides many avenues for exploits, such enclosures should be opened only after due consideration of the inherent risks. The problem lies in the dual roles for which HTML is being used. On the one hand, HTML is surpassing plain, non-tagged ASCII as a common means for composing and exchanging documents. On the other hand, HTML is also being used as an environment to house such things as scripting languages, Java applets, and ActiveX

components. By combining the flexibility to send and receive HTML content with its ability to embody scripts and other forms of programs that have full access to memory and files, the potential for abuse becomes self-evident.

Plug-ins

Plug-ins are native code modules that work in conjunction with software applications to enhance their capabilities. Plug-ins are often added to Web browsers to enable them to support new types of content (audio, video, etc.). Increasingly, plug-ins are also being devised for electronic mail and other desktop software to extend their functionality. Such plug-ins can be downloaded from either the browser vendor's site or a third party site. Browsers typically prompt the user to download a new plug-in when a document that requires functionality beyond the browser's current capabilities. Although plug-ins allow browsers to support new types of content, they are not active content in and of themselves, but simply executables that enable active content technologies. Windows Media player, RealPlayer, ThingViewer, QuickTime, ShockWave and Flash are all examples of plug-ins that allow browsers to support new content types ranging from audio, video, interactive animation, three-dimensional animation effects, and other forms of "new media." Thus, there are two security concerns with plug-ins: the behavior of active content processed by an installed plug-in, and the behavior of the plug-in executables themselves once they are downloaded and installed.

For instance, the ShockWave plug-in from Macromedia provides the ability to render multimedia presentations created in a compatible format, as they are downloaded. By design, ShockWave content supports the Lingo interpretative language as an aid to presentation development. When creating a ShockWave presentation, the author can include custom code using Lingo. Early versions of Lingo allowed the author to make local system calls based on the platform executing the content, potentially allowing malicious code to be downloaded as part of the presentation.

From a security standpoint, plug-ins are executable code and, therefore, precautions should be exercised in obtaining and installing them, as with any other software application. Downloading free software code and authorizing its installation by simply clicking an "Install now" or an equivalent button is risky. Downloading plug-ins directly from a reputable manufacturer is normally less risky, but even in this case, it is difficult for the user to be always aware of the security implications. In the past, unwanted side effects such as changes to browser security settings and tracking of a user's content preferences, however well intentioned, have occurred. Plug-ins designed to animate cursors or hyperlinks have also been designed to track user preferences and viewing habits across a particular Web site more accurately. Although these additional capabilities may improve the user's experience with a particular Web site, the privacy and security implications are often not readily disclosed [Mar00]. Even if the site has a valid identity certificate associated with the signed downloaded code, that only tells the user that the manufacturer of the code has been verified by a certificate authority, but not whether the code obtained from them will behave non-maliciously or correctly. Users of plug-ins should be cautioned to read the fine print before agreeing to download executables, and take adequate measures to backup the system in the event of problems.

Assessing Risk: Using general categories such as high, medium, and low, a practical assessment of technology risks can be made on a qualitative, rather than

quantitative, basis. For example, the following ranking of relative risk, ordered from low to high, can be determined from the previously discussed active content technologies with respect to the execution environment of a browser:

- Portable Document Format
- ShockWave
- JavaScript and VBScript

} Low

- Java Applets
- PostScript
- Visual Basic for Applications

} Medium

- ActiveX Controls

} High

Because these technologies are associated with products that are regularly updated with enhanced features, the relative ranking of a technology may change over time. Moreover, one could legitimately argue moving a technology at the fringe of one category to an adjacent category, based on changing circumstances. For example, because of the ubiquity of JavaScript and VBScript, their high degree of coupling to email clients or other services, and a heightened frequency of detected vulnerabilities, the risk associated with these technologies could be raised from Low to Medium. Similarly, because of Java's extensive security mechanisms and lowered frequency of detected vulnerabilities, the associated risk could be dropped from Medium to Low. Choosing whether a particular active content technology is right for an organization requires a continual balancing of the associated risks against the perceived benefits.

While the focus of this discussion has been mainly on the consumer side of the equation, consideration of the producer side is warranted as well. Often, one technology may be traded off for a lower risk technology without much difficulty. For instance, some ActiveX functionality may be programmed in Java and some Java functionality may be carried out with a scripting language. Similarly, a less troublesome document format may be substituted for another, such as using PDF or Rich Text Format (RTF) in lieu of PostScript or a proprietary word processing file format.

CGI and Related Interfaces

Unlike the above technologies, CGI and other similar server interfaces fall on the producer side of the producer-consumer model. CGI applications can be written in most programming languages to run on a Web server. More often than not, a server-side scripting language such as Perl (Practical extraction and report language) is used for this purpose, because of its flexibility, compactness, and facility. If scripts are not prepared carefully, however, attackers can find and exercise flaws in the code to penetrate a Web server. Therefore, scripts must be written with security in mind and, for example, should not run arbitrary commands on a system or launch insecure programs. An attacker can find flaws through trial and error and does not necessarily need the source code for the script to uncover vulnerabilities.

Two general areas exist where CGI applications can create security vulnerabilities at the server:

- They may intentionally or unintentionally leak information about the host system that can aid an attacker, for example, by allowing access to information outside the areas designated for Web use.

- When processing user-provided input, such as the contents of a form, URL parameters, or a search query, they may be vulnerable to attacks whereby the user tricks the application into executing arbitrary commands supplied in the input stream.

Ideally, server-side scripts should constrain users to a small set of well-defined functionality and validate the size and values of input parameters so that an attacker cannot overrun memory boundaries or piggy back arbitrary commands for execution. In the event that a script does contain flaws, it should be run only with minimal privileges (i.e., non-administrator) to avoid compromising the entire Web site. However, potential security holes can be exploited even when CGI applications run with low privilege settings. For example, a subverted script could have enough privileges to mail out the system password file, examine the network information maps, or launch a login to a high numbered port.

The two areas of vulnerability mentioned potentially affect all Web servers. While these vulnerabilities have frequently occurred with CGI applications, other related interfaces and techniques for developing server applications have not been immune. CGI being an early and well-supported standard has simply gained more notoriety over the years, and the same areas of vulnerability exist when applying similar Web development technologies at the server.

Safeguards

Safeguards are approved security measures taken to prevent or reduce the risk of system compromise. To protect computational resources from attack, appropriate safeguards, such as hardware and software mechanisms, policies, procedures, and physical controls, must be in place. A number of steps can be taken to mitigate the risks in using active content. Overall, there are two main approaches to follow: avoidance – staying completely clear of known and potential vulnerabilities, and harm reduction – applying measures to limit the potential loss due to exposure. The following sections highlight some of the more useful safeguards one can apply.

Security Policy

A security policy is the set of rules, principles, and practices that determine how an organization implements its security. A policy reflects an organization's view on required safeguards, based on a consideration of its assets, the impact of loss or compromise, and the threat environment. Information security in any organization is largely dependent on the quality of the security policy and the processes that an organization imposes on itself. No amount of technology can overcome a badly implemented, poorly planned, or nonexistent security policy. If the policy is not stated clearly and consistently, and not

made known and enforced throughout an organization, it creates a situation ripe for exploitation.

Therefore, having or establishing an organizational security policy is an important first step in applying safeguards for active content. For example, an Internet security policy can address enabling Java, JavaScript, or ActiveX on an individual user's Web browser in various ways:

- Functionality must be disallowed completely.
- Functionality is allowed, but only from internal organizational servers.
- Functionality is allowed, but only from trusted external servers.
- Functionality is allowed from any server.

One practical difficulty is that functionality invariably takes precedence over security in product marketing and consumer demand. Often new technology products are in use within an organization years before the security policy is written to guide employees. For example, the Department of Defense (DoD) has recently completed formulating its policy and guidance on mobile code technology [DoD00]. The policy delineates three categories of technology based on increasing associated risk. Category 1, the most dangerous, includes ActiveX and script languages interpreted at the operating system command level. Category 2 includes Java mobile code, PostScript, and various scripting languages running within the confines of a desktop application. Category 3 includes Shockwave Flash content, PDF, and VBScript and ECMAscript-variant scripting languages interpreted within the confines of a browser. Any mobile code technology not yet assigned to a category by DoD falls into category 1 by default. Where possible, the policy distinguishes between signed and unsigned code, favoring the former over the latter.

While the policy comes many years after the respective technology's debut in products, the DoD started to address the problem early, relative to most other organizations. Yet, on the horizon are technologies, allowing entire processes (i.e., code, accumulated data, and execution state) to move among host computers, which go beyond mobile code and the limits of classical computer security.

Risk Analysis and Management

Security involves continually analyzing and managing risks. Any such analysis must identify vulnerabilities and threats, anticipate potential attacks, assess their likelihood of success, and estimate the potential damage from successful attacks. Experience has shown that use of active content technologies involves risk, since they are frequently accompanied by new vulnerabilities. Risk management is the process of assessing risk, taking steps to reduce risk to an acceptable level, and maintaining that level of risk. One of the most significant security pieces missing from most organizations is an on-going practice of risk analysis and management.

Because security is relative to each organization, it must be tailored to an organization's specific needs, budget, and culture. For example, an attack launched against one organization might succeed easily and compromise extremely important information, but against another organization would only result in minimal damage, perhaps because of an

absence of sensitive data. Companies, much like people, have personalities with differing comfort levels on the amount of risk that is reasonable, which also influence this process. Once an assessment is made, safeguards can be put in place against those attacks deemed significantly high by either reducing the likelihood of occurrence or minimizing the consequences of the attack. Different safeguards are employed to meet an organization's specific needs.

Recently, the General Accounting Office (GAO) analyzed and summarized information security weaknesses identified in audit reports of federal agencies [GAO00], issued from July 1999 through August 2000. They noted that most of the organizations reviewed had not adopted systematic, thorough practices for evaluating system vulnerabilities and for reducing risk.

> "Despite the importance of this aspect of an information security program, poor security planning and management continues to be a widespread problem. As noted earlier, of the 21 agencies for which this aspect of security was reviewed, all had deficiencies. Many of these agencies had not developed security plans for major systems based on risk, had not documented security policies, and had not implemented a program for testing and evaluating the effectiveness of the controls they relied on. As a result, agencies (1) were not fully aware of the information security risks to their operations, (2) had accepted an unknown level of risk by default rather than consciously deciding what level of risk was tolerable, (3) had a false sense of security because they were relying on controls that were not effective, and (4) could not make informed judgments as to whether they were spending too little or too much of their resources on security."

To help remedy the problem, GAO has developed a guide on implementing an information security risk assessment process [GAO99]. It contains examples, or case studies, of practical risk assessment procedures, which have been adopted by several organizations that have successfully established good risk assessment practices. The guide also identifies factors that are important to the success of any risk assessment program, regardless of the specific methodology employed.

Evaluated Information Technology

Where appropriate, consideration should be given to using information technology products that have undergone a formal security evaluation. Products that quarantine or block the behavior of active content are coming onto the market and at least one has undergone formal evaluation. The Evaluation Criteria for Information Technology (IT) Security, more commonly known as the Common Criteria (CC), is the prevailing international standard for specifying and evaluating the security features of computer products and systems [IS15408]. The National Information Assurance Partnership (NIAP), jointly sponsored by NIST and NSA, has responsibility for CC related activities within the US. The focus of a CC security evaluation is primarily on the correctness and effectiveness of the design, under the well-founded principle that a sound design enables a secure implementation, but an unsound design is hopelessly doomed. Products that have undergone other, less formal forms of third-party testing and evaluation also merit consideration over those lacking such scrutiny. However, an evaluated product absent a needed security capability might be less preferable to a product having a lower level of assurance that offers such a capability. A more detailed recommendation for Federal

Organizations on the acquisition and use of evaluated and tested products is available elsewhere [Rob00].

Note that using tested and evaluated software does not necessarily ensure a secure operational environment. The way in which a product is applied and composed with other system components affects security. Even when a product does successfully complete a formal security evaluation, it may contain vulnerabilities. For example, one of the most common attacks, if not the most common, is through a buffer overflow, whereby the input to a defined programming interface is carefully crafted to overwrite memory beyond the input buffer limit with instructions designed to gain control of the process. Most people would probably expect a security evaluation to include a systematic search and elimination of buffer overflows. Unfortunately, it does not. While evaluators test the implementation for known security vulnerabilities, and at more stringent levels even attempt penetrations, a systematic search of buffer overflow vulnerabilities is out of scope due mainly to cost. Automatic discovery of buffer overflow vulnerabilities within code is a research challenge.

Security Audit

An increasing popular approach for measuring the security posture of an organization is through a formal security audit. Audits ensure that policies and controls already implemented are operating correctly and effectively. Audits can include static analysis of policies, procedures, and safeguards as well as active probing of the systems external and internal security mechanisms. The results of an audit identify the strengths and weaknesses of the security of the system and provide a list of noted deficits for resolution, typically ranked by degree of severity. Because the security posture of a system evolves over time, audits are most effective when done on a reoccurring basis.

While periodic formal audits are useful, they are not a replacement for day-to-day management of the security status of a system. Enabling system logs and reviewing their contents manually or through automated reports summaries can sometimes be the best means of uncovering unauthorized behavior and detecting security problems. A well-publicized example of this is documented in Cliff Stoll's book, *The Cuckoo's Egg*, where a 75-cent accounting error appearing in a computer log eventually leads to the discovery of an industrial espionage ring. Some security product manufacturers' sites, such as finjan software's site (http://www.finjan.com/mcrc/test.cfm), offer the ability to launch non-malicious examples of common attacks involving active content, to test the security posture of a personal computer.

Application Settings

The desktop applications that handle active content documents typically have built-in controls that can be used to control or prevent access. For example, both Netscape and Microsoft Web browsers have options or preferences menus that can be used to select appropriate security settings regarding downloadable active content. The National Security Telecommunications and Information Systems Security Committee (NSTISSC) has recently issued an advisory memorandum on Web browser security [NAM00] that outlines steps to lower associated risks through tightly controlled configurations. Electronic mail, spreadsheet, word processor, database, and presentation graphic desktop software applications have control settings similar to those of a browser and demand scrutiny in light of past exploits. For example, the ability of many electronic mail applications to render HTML formatted content can be controlled to disallow or disable

any executable content. Tight functional binding among desktop applications is a concern, particularly where automatic rendering of multi-part or composite documents is enabled. Even today, products are delivered with insecure default settings. It behooves the user of such applications to become familiar with the security options available and use them in accordance with organizational policy.

Version Control

Administrators of systems can gain better security by routinely applying security patches when they become available. This is a well-known and effective remedy, but for a variety of reasons also a well-ignored one. Users and administrators can also take advantage of security enhancements to applications that they manage by upgrading to newer versions, when appropriate. Updating software products automatically over the Web is becoming increasingly popular, as the benefits are considerable. For example, users of Microsoft Windows 98 and above can use a built-in Windows update feature to find bug fixes and product updates and download them automatically from the Web. Using this feature, however, requires the downloading of an ActiveX control that scans the computer for any needed updates particular to software already installed. Microsoft's Web site states that none of this information is sent over the Internet or back to Microsoft. As this practice becomes more commonplace, users must be aware of their implicit decision to allow a vendor to run software on their machine and act accordingly, following prescribed policy.

Incident Response Handling

No matter how well an organization's security program is executed, inevitably, a security breach will occur in a system. Besides adopting reasonable precautions for securing computer systems and networks, one must also establish the ability to respond quickly and efficiently when a security incident occurs. A security incident is an adverse event or situation involving a networked information system that poses a threat to the integrity, confidentiality, or availability of computational resources. Examples of incidents include unauthorized use of an account, unauthorized elevation of system privileges, and execution of malicious code that corrupts data or other code. Incidents may result in a partial or complete loss of security controls, an attempted or actual compromise of data, or the waste, fraud, abuse, loss or damage of computational resources. Active content has played an increasingly significant role in security incidents. Responding to computer security incidents effectively requires a significant amount of preparation. Incident response activities require technical knowledge as well as effective communication and coordination among personnel who respond to the incident, in order to return the system as quickly as possible to normal operations. Proper periodic backup of critical files is a key ingredient in recovering from the adverse effects of an incident.

Automated Filters

Once forms of malicious content have been identified and understood, they can be selectively detected and eliminated or completely blocked from entry. For example, firewalls can be augmented to filter certain types of electronic mail attachments and Web content that have known malicious code characteristics, and reject them at a point of entry. Anti-virus software has also become increasingly capable of detecting malicious code signatures within active content.

Besides ingress filtering, egress filtering is also useful in denying unacceptable actions originating from internal hosts. Strange or unexpected, but not necessarily unacceptable, transmissions from internal hosts may signal that they have been compromised in some way. Intrusion detection systems provide an additional safeguard for screening network and host behavior and provide notification when either inappropriate or unusual event sequences occur, or signatures of known exploits are matched.

While firewalls, anti-virus software, and intrusion detection tools provide useful safeguards, they are not foolproof. Constructing a program to detect with certainty the presence or absence of harmful code within arbitrary programs or protocol is impossible. Moreover, a variety of techniques exist for deception such as mutation, segmentation, and disguise via extended character set encoding. Thus, screening tools are faced with the prospect of diminishing returns – greater investments are needed for small increases in effectiveness. Despite services to refresh protection software with signatures of known exploits, and cascaded defense-in-depth measures, apart from total isolation, there is no guarantee that something harmful cannot get through.

Behavioral Controls

While nearly all interpretive execution environments, such as those supported by Web browsers and desktop applications, impose restrictions on the languages they process and execute, those restrictions may not be sufficient for all organizations. For such situations, imposing additional or redundant restraints on the code's behavior as it executes (e.g., privilege or function) might be appropriate. Conceptually, behavior controls can be viewed as a software cage or quarantine mechanism that dynamically intercepts and thwarts attempts by the subject code to take unacceptable actions that violate policy. As with firewall and anti-virus products, technologies that dynamically restrain mobile code were born out of necessity to supplement existing mechanisms, and represent an emerging class of security product. Such products are intended to complement firewall and anti-virus products that respectively block network transactions or mobile code based on predefined signatures (i.e., content inspection), and may refer to technologies such as dynamic sandbox, dynamic monitors, and behavior monitors, used for controlling the behavior of mobile code [Vib01]. In addition to mobile code, this class of product may also be applicable to stationary code or downloaded code whose trustworthiness is in doubt.

Readers

Occasionally, manufacturers of desktop applications provide free software readers, which can interpret their proprietary file formats for document recipients who do not own the application. The Adobe Acrobat Reader, for example, allows users to view and print PDF files, but does not allow users to create or edit them. Since software readers are only intended to produce a viewable rendition of the document and have limited inherent capabilities, they bypass many potentially harmful features and exploits based on implementation vulnerabilities contained in the full-fledged application. Besides manufacturer-provided readers, general-purpose software readers are commercially available, which can render dozens of different file formats.

A related measure is the selection of less capable types of active content documents, when multiple choices are offered. Some Web sites offer an electronic document in a variety of formats such as proprietary word processor format, PostScript, or PDF. While PDF

represents text and graphics using the PostScript language-imaging model, PDF is not a programming language and does not support macros, making it the safest alternative. Whenever possible, content providers and site operators should strive to provide material encoded in less harmful document formats. For example, when document distillers are not available to convert textual documents into PDF, a good alternative is to make available a version in ".rtf", rich text format, rather than a proprietary word processing format.

Digital Signature

A digital signature is an unforgeable code computed over a document or other information that uniquely identifies the signer who computed it. When applied properly, a digital signature serves as a means of confirming the authenticity of an object, its origin, and its integrity. Because of these characteristics, digital signatures are involved in most authentication schemes. For example, the Secure Sockets Layer (SSL) protocol, built into most browsers and Web servers, relies on digital signatures for authenticating the parties involved in a transaction. When applied to mobile code, the code signer is typically the individual or organization that created the code.

Digital signatures involve public key cryptography, which relies on a pair of keys associated with an entity. One key is kept private by the signing entity and the other is made publicly available. Passing mobile code through a non-reversible hash function provides a fingerprint or message digest of the code. Applying a digital signature function to the message digest using the private key of the signer forms a digital signature. A recipient uses the signer's public key to verify the signature conveyed with the code. The digital signature is in effect an integrity mechanism, since changes to the code would invalidate the signature and, thus, be detectable by the recipient. Digital signatures benefit greatly from the availability of a Public Key Infrastructure, since certificates containing the identity of an entity and its public key (i.e., a public key certificate) can be readily located and verified. This allows the code, signature, and public key certificate to be forwarded to a recipient, who can easily verify the source and authenticity of the code.

When appropriate, digital signatures should be considered for use in applications involving active content, to not only verify the identities of the various parties involved, but also confirm the integrity of any mobile code and the acceptability of the code's author. Note that the meaning of a signature may be different depending on the policy associated with the signature scheme and the party who signs. For example, the author of some code, either an individual or organization, may use a digital signature to indicate who produced the code, but not to guarantee that the agent performs without fault or error. In fact, author-oriented signature schemes, such as Microsoft's Authenticode, were originally intended to serve as digital shrink wrap, whereby the original product warranty limitations stated in the license remain in effect (e.g., the manufacturer makes no warranties as to the fitness of the product for any particular purpose). Many users, however, misinterpret the significance of such a signature scheme beyond its original intent of establishing the authenticity of distributed software. Instead, for them it has become a form of trust in the software's behavior, which could ultimately have disastrous consequences.

Isolation

Isolation can be applied at various levels. The simplest is complete isolation at the system level. A production computer system that is unable to receive active content documents

cannot be affected by malicious hidden code. Although isolating a system physically is not always possible, logical isolation (e.g., via router settings or firewall controls) may be applied, at least partially. For example, risky functions, such as Web browsing, may be restricted to trusted sites only or confined to a second system designated and configured exclusively for that purpose. Often older or spare systems are available, which could be put to good use this way.

Isolating tightly bounded, proprietary program components is another alternative. Seamless interoperation of products such as electronic mail, Web browsers, and office applications is a goal of product manufacturers. To provide better functionality or performance, manufacturers often allow products within their product line to take advantage of little known or undocumented programming interfaces, which from time-to-time has lead to unwanted or insecure side effects. By integrating products from different manufacturers, one can effectively isolate program components from using all but the standard documented interfaces.

Minimal Functionality

Security is inversely related to complexity – the more complex a system, the more difficult it is to secure. Prudent users and administrators should remove unnecessary applications and program components to reduce complexity and shut off possible avenues of attack. Though a system configuration may have a function logically disabled, a clever attacker may be able to alter the settings to enable the functionality and then use it in an exploit. On the browser side, unnecessary plug-ins or ActiveX controls should be removed. On the server side, any unnecessary software not needed in providing Web services must go as well, particularly any development tools that could be used to further an attack, should an intruder gain an initial foothold.

Least Privilege

The principle of least privilege states that programs should operate only with the privileges needed to accomplish their functions. During application development, it is easier to run code at the highest level, with the intention of paring back privileges in the production deployment. Unfortunately, the privilege reduction step is easy to overlook and often is. For example, Unix developers may enhance the server using Set-User-ID (SUID) programs, which refer to code that run with privileges of the owner (e.g., root) regardless of who is executing them. SUID programs, particularly those owned by root, can be dangerous because if subverted, they allow an intruder to gain control with the owner's privilege. Running the code instead with the minimum privileges needed, restricts the range of access to the intruder, if an attack is successful. Similarly, on the browser side, any mobile code received should be constrained to the minimal privileges needed. The recent versions of the Java Virtual Machine environment, for example, offer the user the ability to set fine-grained permission controls for incoming applets.

Layered and Diverse Defenses

Defending an information system requires safeguards to be applied not only at points of entry, but also throughout it. Ideally, the selection and placement of security controls are done in such a way that all attacks are progressively weakened and eventually defeated. Having an identical control in succession tends to only lengthen the duration of the attack.

Applying different types of controls that complement each other and are mutually supportive is a much more effective approach. While the capabilities of available safeguards may overlap to some extent, their combined effect should be far greater than the sum of each individual effect. For example, if control A misses 30% of attacks and control B also misses 30%, in combination they should only miss about 9% (.3 x .3) of attacks. The previous sections mention a number of complementary safeguards that are available, including classes of products that employ firewall, anti-virus, intrusion detection, and behavior blocking technologies.

Summary

Active content documents offer benefits to both the consumers and producers of such documents. The associated technologies are varied, yet sometimes similar and overlapping in function. Java applets, JavaScript, VBScript and ActiveX provide additional functionality to Web pages, while plug-ins, helper applications, and ActiveX controls enable browsers to support new types of content. PostScript offloads the processing and interpretation of the presentation of documents to the printer or display interpreter, and macros automate repetitive word processing and spreadsheet tasks. HTML, JavaScript, and Java are relatively platform independent and can run on current versions of both Internet Explorer (IE) and Netscape Navigator. VBScript and JavaScript can also be used to pass information between HTML, Java, and ActiveX components.

The benefits of each of these active content technologies must be carefully weighed against the risks they pose. Security is not black or white, but shades of gray. When employing active content technology, security measures should be put in place to reduce risk to an acceptable level and to recover if an incident occurs.

Informed security officers, administrators, and other IT professionals are responsible for developing security policies based on their organization's specific security needs and level of acceptable risk. Unfortunately, rarely is there a "one size fits all" guideline that fits the unique needs of every organization. Thus, each organization must decide for itself what constitutes an acceptable level of risk and act accordingly. Establishing an organizational security policy is an important step in developing and applying appropriate security measures. The IT and security staff have a responsibility for keeping abreast of the risks associated with emerging technologies, by subscribing to security mailing lists and visiting vendor Web sites for information and updates to products used within their organization. As active content moves beyond desktop personal computers to mobile handheld or wearable devices, television sets, and other consumer electronic goods, users will be faced with competing and difficult tradeoffs of decreased privacy and security against increased functionality and ease-of-use.

Before handling active content documents, consider seriously the following checklist, which summarizes some recommendations drawn from the material presented in the previous sections:

- Develop (or follow) the enterprise security policy regarding active content.
- Identify and assess the risk to critical information resources from active content.

- Audit systems on a regular basis to ensure the security policy is implemented correctly and remains effective.
- Identify critical information resources and maintain regular backups.
- Become knowledgeable of the security settings of desktop applications and turn off unneeded functionality.
- Keep systems current with the latest software upgrades and patches that address security vulnerabilities in desktop applications, such as Web browsers, readers, and electronic mail, and other critical software.
- Obtain all software through approved distribution channels.
- Evaluate and install virus scanners, firewalls, active content filters, and dynamic behavior monitors according to enterprise security requirements. Keep these products upgraded to the latest version.
- Read the fine print before agreeing to download application software and plug-ins.
- Institutionalize how needed plug-ins and freeware are obtained from software manufactures, evaluated, and distributed throughout the organization.
- Do not peruse active content or run downloaded software from untrusted sources. Enable ActiveX code only from trusted Web sites that require its use.
- Create and distribute active content documents only after carefully considering the risk and benefits. Where possible, use forms of active content that poses the least risk
- Consider using an isolated system and safe browser settings when visiting untrusted Web sites.
- Limit the applications installed on a system, deleting any that are not used or no longer needed.
- Disable JavaScript and any other active content processing capabilities within electronic mail desktop applications that are capable of handling HTML or other mark up language encoded messages.
- Do not open active content documents or execute any electronic mail attachments, without first verifying them with the sender. Be especially wary of attachments to electronic chain mails forwarded from or through friends.
- Keep informed of latest security advisories from either the Federal Computer Incident Response Center (FedCIRC) or the Computer Emergency Response Team (CERT) Coordination Center, and subscribe to security mailing lists.
- Periodically crosscheck products against published lists of known vulnerabilities, such as the NIST ICAT vulnerability database, which provides pointers to solution resources and patch information.
- Regularly audit systems and networks, quickly remedying any deficits noted.

- Know who to contact and what steps to take when discovering evidence of an intrusion.

One typical and common sense approach is to improve the security infrastructure incrementally over time. At each step, apply safeguards against the most critical risk items. For example, start inexpensively with firewalls and gateway servers capable of screening active content and executable electronic mail attachments, and successfully defending against a high percentage of Internet launched attacks. Later, for additional protection, complement anti-virus software with behavioral controls, intrusion detection capabilities, or other technologies. Regular site security audits also help to identify vulnerabilities and appropriate safeguards, and to decide whether the remaining risks warrant further expenditures of time and money.

Terminology

The following definitions highlight important concepts used throughout this document:

Active Content

Active content refers to electronic documents that can carry out or trigger actions automatically on a computer platform without the intervention of a user. Active content technologies allow mobile code associated with a document to execute as the document is rendered.

Buffer Overflow

As the name implies, a buffer overflow is a condition at an interface under which more input can be placed into a buffer or data holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system.

Computer Virus

A computer virus is similar to a Trojan horse insofar as it is a program that hides within a program or data file and performs some unwanted function when activated. The main difference is that a virus can replicate by attaching a copy of itself to other programs or files, and may trigger an additional "payload" when specific conditions are met.

Cookie

Cookies entail a piece of state information supplied by a Web server to a browser, in a response for a requested resource, for the browser to store temporarily and return to the server on any subsequent visits or requests.

Easter Egg

An Easter egg is hidden functionality within an application program, which becomes activated when an undocumented, and often convoluted, set of commands and keystrokes

is entered. Easter eggs are typically used to display the credits for the development team and intended to be non-threatening.

Interpreter

An interpreter is a program that processes a script or other program expression and carries out the requested action, in accordance with the language definition.

Macro Virus

A macro virus is a specific type of computer virus that is encoded as a macro embedded in some document and activated when the document is handled. Many desktop applications, such as word processors and spreadsheets, support powerful macro languages that can be exploited this way.

Malicious Code

Malicious code refers to programs that are written intentionally to carry out annoying or harmful actions. They often masquerade as useful programs or are embedded into useful programs, so that users are induced into activating them. Types of malicious code include Trojan horses, computer viruses, and worms.

Mobile Code

Mobile code refers to programs (e.g., script, macro, or other portable instruction) that can be shipped unchanged to a heterogeneous collection of platforms and execute with identical semantics. The term also applies to situations involving a large homogeneous collection of platforms (e.g., Microsoft Windows).

Risk

Risk is measure of the likelihood and the consequence of events or acts that could cause a system compromise, including the unauthorized disclosure, destruction, removal, modification, or interruption of system assets.

Safeguards

Safeguards are approved security measures taken to protect computational resources by eliminating or reducing the risk to a system. Available safeguards include hardware and software mechanisms, policies, procedures, and physical controls. Safeguards, put in place to counter a specific threat or attack, are also known as countermeasures.

Script

A script is a sequence of instructions, ranging from a simple list of operating system commands to full-blown programming language statements, which can be executed automatically by an interpreter. Unlike compiled programs, whose instructions are native to a class of computer processor, a script involves portable instructions that must be processed by an “interpreter” program to carry out the indicated action.

Scripting Language

A scripting language defines the syntax and semantics for writing scripts. Typically, scripting languages follow the conventions of a simple programming language, but they can also take on a more basic form such as a macro or a batch file. JavaScript, VBScript, Tcl, Perl, and PHP are examples of scripting languages.

Spyware

Spyware is a program embedded within an application that collects information and periodically communicates back to its home site, unbeknownst to the user. Spyware programs have been discovered within shareware or freeware programs, without notification of this hidden functionality given in the license agreement or elsewhere.

Threat

Threats are possible dangers to a computer system, which may result in the interception, alteration, obstruction, or destruction of computational resources, or in some other way disrupt the system.

Trojan Horse

A Trojan horse is a useful or seemingly useful program that contains hidden code of a malicious nature. When the program is invoked, so is the undesired function whose effects may not become immediately obvious. The name stems from an ancient exploit of invaders' gaining entry to the city of Troy by concealing themselves in the body of a hollow wooden horse, presumed to be left behind by the invaders as a gift to the city.

Vulnerability

Vulnerabilities are flaws or weaknesses in a computer system, its security procedures, internal controls, or design and implementation, which could be exploited to violate the system security policy.

Web Browser

A browser refers to any collection of software that lets individuals view Web content, and includes the user interface, helper applications, language and byte code interpreters, and other similar program components.

Web Bug

Web bugs are tiny images, invisible to a user, placed on Web pages in such a way to enable third parties to track use of Web servers and collect information about the user, including IP address, host name, browser type and version, operating system name and version, and cookies.

Worm

A worm is a self-replicating program that propagates itself through a network onto other computer systems. Unlike a virus, it is self-contained and does not require a host program

or any user intervention to replicate. Although worms are associated with malicious code nowadays, the concept was introduced originally as a means of building useful applications [Sho82].

On-line Resources

A wealth of security information, which supplements this publication, is available on-line. Note that, in addition to this section, many of the publications in the reference section contain URLs for perusal by the reader. The following list of Web sites contains a number of notable government, industry, and university sites where one can begin to explore additional information on computer security.

NIST ICAT Vulnerability Database

ICAT is a searchable index of information on computer vulnerabilities. It provides a manufacturer and product oriented search capability at a fine granularity, and links users to vulnerability and patch information. http://icat.nist.gov

NIST Computer Security Resource Clearinghouse (CSRC)

The CRSC contains current US security policy documents, calendar of events, security publications, training resources, and information on various computer security subjects. http://csrc.nist.gov

FBI National Infrastructure Protection Center (NIPC)

NIPC serves as the U.S. government's focal point for threat assessment, warning, investigation, and response for threats or attacks against the nation's critical infrastructures, which include telecommunications, energy, banking and finance, water systems, government operations, and emergency services. http://www.nipc.gov

National Information Assurance Partnership (NIAP):

NIAP is a U.S. Government initiative to promote the development of technically sound security requirements for IT products and systems and appropriate metrics for evaluating those products and systems to meet the needs of both information technology (IT) producers and consumers. http://www.niap.nist.gov/

Federal Computer Incident Response Center (FedCIRC).

FedCIRC provides a government focal point for incident reporting, handling, prevention, and recognition. http://www.fedcirc.gov/

Computer Emergency Response Team (CERT) Coordination Center

CERT issues security advisories, helps start other incident response teams, coordinates the efforts of teams when responding to large-scale incidents, provides training to incident response professionals, and researches the causes of security vulnerabilities. http://www.cert.org/

WWW Consortium Security FAQ

The World Wide Web Consortium site contains a repository of information about the Web for developers and users. http://www.w3.org/Security/Faq/

Center for Education and Research in Information Assurance and Security (CERIAS):

CERIAS is a university center for multidisciplinary research and education in areas of information security (computer security, network security, and communications security), and information assurance. http://www.cerias.purdue.edu/

RISKS forum

ACM Committee on Computers and Public Policy forum advises on risks to the public in computers and related systems. http://catless.ncl.ac.uk/Risks/

Microsoft Internet Explorer Security Page

Microsoft posts information and code fixes for security problems in their products as soon as the information is available. http://www.microsoft.com/windows/ie/security/default.asp

Netscape Security Page

Netscape posts the latest news concerning the security of their browser, Web server, and development software. http://home.netscape.com/security/notes/

Sun Microsystems Java Security Page

Sum maintains the latest information Java Security, including answers to frequently asked questions, white papers, and articles. http://java.sun.com/security/

System Administration, Networking, and Security (SANS) Institute

The SANS community offers various types of products and services, including: system and security alerts, news updates, special research projects and publications, in-depth education, and certification. http://www.sans.org

References

[Ado00a] Adobe Portable Document Format, Version 1.3, second edition, Adobe Systems Incorporated, July 2000, http://partners.adobe.com/asn/developer/acrosdk/DOCS/PDFRef.pdf

[Ado00b] “Security Update,” Adobe Systems Incorporated, 2000, http://www.adobe.com/misc/pdfsecurity.html

[Ado99] PostScript Language Reference, third edition, Adobe Systems Incorporated, February 1999, http://partners.adobe.com/asn/developer/PDFS/TN/PLRM.pdf

[Anu98] V. Anupam, A. Mayer, "Secure Web Scripting," IEEE Internet Computing, vol. 2, no. 6, November/December 1998, pp. 46-55

[Bec99] K. Beck, "Embracing Change with Extreme Programming," IEEE Computer, vol. 32, issue 10, Oct. 1999, pp. 70 -77

[Cle90] Robert E. Van Cleef, "New Roque Imperils Printers," The Risks Digest, Volume 10, Issue 32, September 1990, http://catless.ncl.ac.uk/Risks/10.32.html

[Coh95] Fred Cohen, "Internet Holes: 50 Ways to Attack Your Web Systems," Management Analytics, 1995, http://all.net/journal/netsec/9512.html

[CNET97] "ActiveX Used as Hacking Tool," CNET News.com, February 7, 1997, http://news.cnet.com/news/0-1005-200-316425.html?tag=st.ne.ni.rnbot.rn.ni

[CSS1] Cascading Style Sheets – level 1 (CSS1), W3C Recommendation, January 1999, http://www.w3.org/TR/1999/REC-CSS1-19990111

[CSS2] Cascading Style Sheets – level 2, (CSS2), W3C Recommendation, May1998, http://www.w3.org/TR/1998/REC-CSS2-19980512

[Cus99] Michael A. Cusumano, David B. Yoffie, "Software Development on Internet Time," IEEE Computer, October 1999, pp.60-69

[Cza95] G. Czajkowski, T. von Eicken, "JRes: A Resource Accounting Interface for Java," ACM Conference on Object Oriented Languages and Systems (OOPSLA), Vancouver, Canada, October 1998

[DoD00] "Policy Guidance for Use of Mobile Code Technologies in Department of Defense (DoD) Information Systems," Assistant Secretary of Defense Memorandum, November 7, 2000, http://www.c3i.osd.mil/org/cio/doc/mobile-code11-7-00.html

[ECMA99] ECMAScript Language Specification, 3rd edition, Standard ECMA-262, December 1999, ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf

[Fel97] Edward W. Felten et ali, "Web Spoofing: An Internet Con Game," National Information Systems Security Conference, October 1997, http://www.cs.princeton.edu/sip/pub/spoofing.html

[Fis01] Dennis Fisher, "'Peachy' Virus Targets PDF Files," Interactive Week, August 8, 2001, http://www.zdnet.com/intweek/stories/news/0,4164,2802558,00.html

[FTP] Postel, J., Reynolds, J., File Transfer Protocol (FTP), IETF Network Working Group, STD 9, RFC 959, October 1985, http://www.ietf.org/rfc/rfc0959.txt?number=959

[Fug98] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," IEEE Transactions on Software Engineering, 24(5), May 1998, pp. 342-361, http://www.cs.ucsb.edu/~vigna/listpub.html

[GAO00] "Information Security: Serious and Widespread Weaknesses Persist at Federal Agencies," United States General Accounting Office (GAO), GAO/AIMD-00-295, September 2000, http://www.gao.gov/new.items/ai00295.pdf

[GAO99] "Information Security Risk Assessment: Practices of Leading Organizations," United States General Accounting Office (GAO), GAO/AIMD-00-33, November 1999, http://www.gao.gov/special.pubs/ai00033.pdf

[Gil97] John Gilles, "Crackers Shuffle Cash with Quicken, ActiveX," Wired News, February 7, 1997, http://www.wirednews.com/news/technology/0,1282,1943,00.html

[Gon98] L. Gong, "Java Security Architecture (JDK 1.2)," version 1.0, Sun Microsystems, October 1998, http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html

[Gos96] J. Gosling, H. McGilton, "The Java Language Environment: A White Paper," Sun Microsystems, May 1996, http://SunSITE.sut.ac.jp/java/whitepaper/

[Hir99] Shane Hird, "Adobe Acrobat Viewer ActiveX Buffer Overflow Vulnerability," September 27, 1999, http://www.securityfocus.com/bid/666

[HTML4] HTML 4.01 Specification, W3C Recommendation, December 1999, http://www.w3.org/TR/html4/

[HTTP] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, and T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, IETF Network Working Group, RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt?number=2616

[Hug99] Paul Hughes, "Building a Web Browser," THT Productions Inc., December 1999

[IS15408] Evaluation Criteria for Information Technology Security, ISO/IEC International Standard (IS) 15408, Parts 1 thru 3, JTC 1/SC 27, June 1999

[Mar00] David Martin Jr., Richard Smith, Michael Brittain, Ivan Fetch, Hailin Wu, "The Privacy Practices of Web Browser Extensions," Privacy Foundation, December 6, 2000, http://www.privacyfoundation.org/pdf/bea.pdf

[MIME] N.Borenstein and N. Freed, MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, IETF Network Working Group, RFC 1521, September 1993, http://www.ietf.org/rfc/rfc1521.txt?number=1521

[Mor98] John F. Morar, David M. Chess, "Web Browsers – Threat or Menace?" Virus Bulletin Conference, October 1998, http://www.research.ibm.com/antivirus/SciPapers/Chess/Threat/Threat.html

[NAM00] "Advisory Memorandum on Web Browser Security Vulnerabilities," NSTISSAM INFOSEC 3-00, August 2000, http://csrc.nist.gov/publications/secpubs/index.html#other

[Rob00] Ed Roback, "Guidelines to Federal Organizations on Security Assurance and Acquisition/Use of Tested/Evaluated Products," NIST SP 800-23, August 2000, http://csrc.nist.gov/publications/nistpubs/800-23/sp800-23.pdf

[Sha99] Stephen Shankland, "Melissa's Mischief Hits All Sides," CNET News.com, March 31, 1999, http://news.cnet.com/news/0-1005-200-340611.html?tag=st.ne.ni.rnbot.rn.ni

[Sho82] Shoch, John F., and Jon A. Hupp, "The Worm Programs – Early Experience with a Distributed Computation," Communications of the ACM Volume 25, Number 3, March 1982, pp. 172-180

[Spe90] Henry Spencer, "Re: New Roque Imperils Printers," The Risks Digest, Volume 10, Issue 35, September 1990, http://catless.ncl.ac.uk/Risks/10.35.html

[Ste00] Lincoln D. Stein, "The World Wide Web Security FAQ," Version 2.0.1, March 2000, http://www.w3.org/Security/Faq/www-security-faq.html

[SUN01] Chronology of Security-related Bugs and Issues, Java Security, Sun Microsystems, February 2001, http://java.sun.com/sfaq/chronology.html

[Ven99] Venners, Bill, "A Walk through Cyberspace," JavaWorld, December 1999, http://www.javaworld.com/javaworld/jw-12-1999/jw-12-jiniology.html

[Ver01] VeriSign Security Alert – Fraud Detected in Authenticode Code Signing Certificates, March 22, 20001, http://www.verisign.com/developer/notice/authenticode/

[Vib01] Robert Vibert, "AV Alternatives: Extending Scanner Range," Information Security Magazine, February 2001, http://www.infosecuritymag.com/articles/february01/features_av_alternatives.shtml

[Wer99] Werring, Laurentius, " The Hidden Threat Within," 11th Annual Canadian Information Technology Security Symposium, May 1999, pp. 201-214

[WML] Wireless Application Protocol (WAP) Wireless Markup Language Specification, WAP-191, Version 1.3, WAP Forum, February 19, 2000, http://www1.wapforum.org/tech/documents/WAP-191-WML-20000219-a.pdf

[XML1] Extensible Markup Language (XML) 1.0, Second Edition, W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

Annex A – HTTP Request Methods

- **Table 1: Summary of Available Browser Request Methods**

Method	Class	Meaning
OPTIONS	Probe	Get information about the communication options available
GET	Retrieval	Retrieve the resource identified by URL
HEAD	Probe	Retrieve meta-information (not content) about the identified resource
POST	Storage	Send data to the server
PUT	Create/Replacement	Send data to the server
DELETE	Removal	Delete the identified resource
TRACE	Diagnostic	Loop back this message
CONNECT	Server Error	Reserved for SSL tunneling via a proxy

Annex B – HTTP Response Status

- **Table 2: Categories of Server Response Code**

Status Code	Class	Meaning
1xx	Informational	Request was received; continuing process
2xx	Success	The action was successfully received, understood, and accepted
3xx	Redirection	Further action must be taken in order to complete the request
4xx	Client Error	The request contains bad syntax or cannot be fulfilled
5xx	Server Error	The server failed to fulfill an apparently valid request