Fortify Security Report

2024-6-21

ASUS

Executive Summary

Issues Overview

On 2024-6-21, a source code review was performed over the civetweb code base. 147 files, 7,339 LOC (Executable) were scanned and reviewed for defects that could lead to potential security vulnerabilities. A total of 7 reviewed findings were uncovered during the analysis.

Issues by Fortify Priority Order	
Critical	6
High	1

Recommendations and Conclusions

The Issues Category section provides Fortify recommendations for addressing issues at a generic level. The recommendations for specific fixes can be extrapolated from those generic recommendations by the development group.

Project Summary

Code Base Summary

Code location: E:/workspace/repos/civetweb
Number of Files: 147
Lines of Code: 7339
Build Label: <No Build Label>

Scan Information

Scan time: 01:25
SCA Engine version: 20.1.1.0007
Machine Name: DESKTOP-MK5UPFE
Username running scan: ASUS

Results Certification

Results Certification Valid

Details:

Results Signature:

SCA Analysis Results has Valid signature

Rules Signature:

There were no custom rules used in this scan

Attack Surface

Attack Surface:
Command Line Arguments:
null.null.null

File System:
null.null.open
null.file.read

System Information:
null.null.DebugProtocolParser
null.null.handleHeapDumpGet
null.null.~file_function
os.null.listdir

Filter Set Summary

Current Enabled Filter Set:
Quick View

Filter Set Details:

Folder Filters:
If [fortify priority order] contains critical Then set folder to Critical
If [fortify priority order] contains high Then set folder to High
If [fortify priority order] contains medium Then set folder to Medium
If [fortify priority order] contains low Then set folder to Low
Visibility Filters:
If impact is not in range [2.5, 5.0] Then hide issue
If likelihood is not in range (1.0, 5.0] Then hide issue

Audit Guide Summary

J2EE Bad Practices

Hide warnings about J2EE bad practices.
Depending on whether your application is a J2EE application, J2EE bad practice warnings may or may not apply. AuditGuide can hide J2EE bad practice warnings.
Enable if J2EE bad practice warnings do not apply to your application because it is not a J2EE application.

Filters:
If category contains j2ee Then hide issue
If category is race condition: static database connection Then hide issue

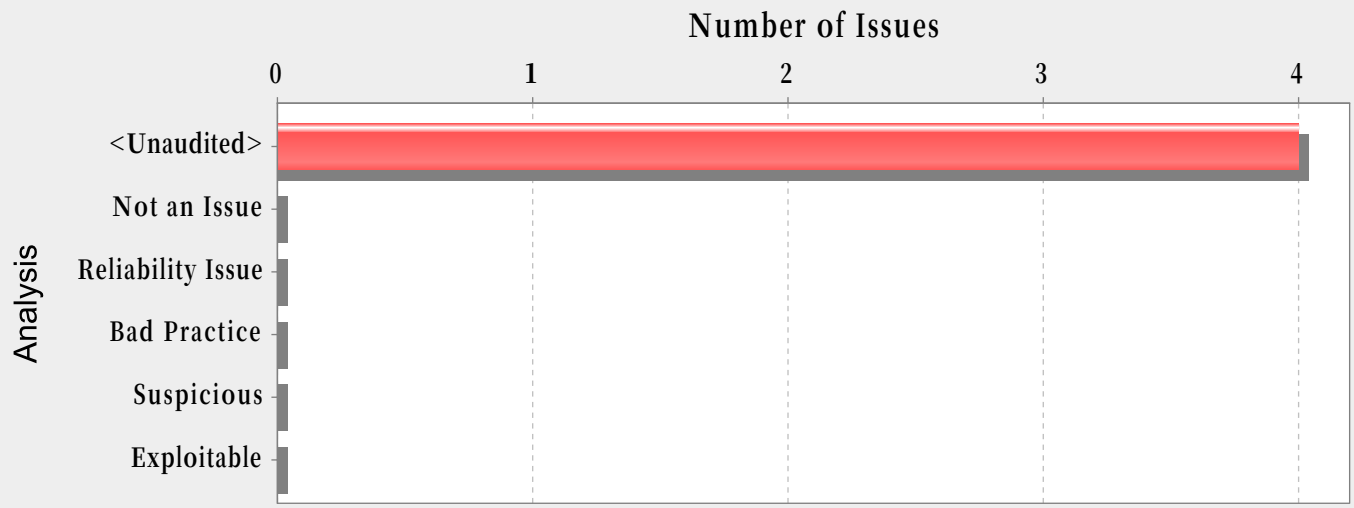
Results Outline

Overall number of results

The scan found 7 issues.

Vulnerability Examples by Category

Category: Insecure Transport (4 Issues)



Abstract:
调用 duk_debug.js 中第 1847 行的 get() 会使用未加密的协议（而非加密的协议）与服务器通信。

Explanation:
所有利用 HTTP、FTP 或 Gopher 的通信均未经过身份验证和加密。因此可能面临风险，特别是在移动环境中，设备要利用 WiFi 连接来频繁连接不安全的公共无线网络。

示例 1：以下示例通过 HTTP 协议（而不是 HTTPS 协议）读取数据。

```
var http = require('http');
...
http.request(options, function(res){
...
});
...
```

由于传入的 http.IncomingMessage 对象 res 通过未加密和未经验证的通道传输，因而可能存在安全隐患。

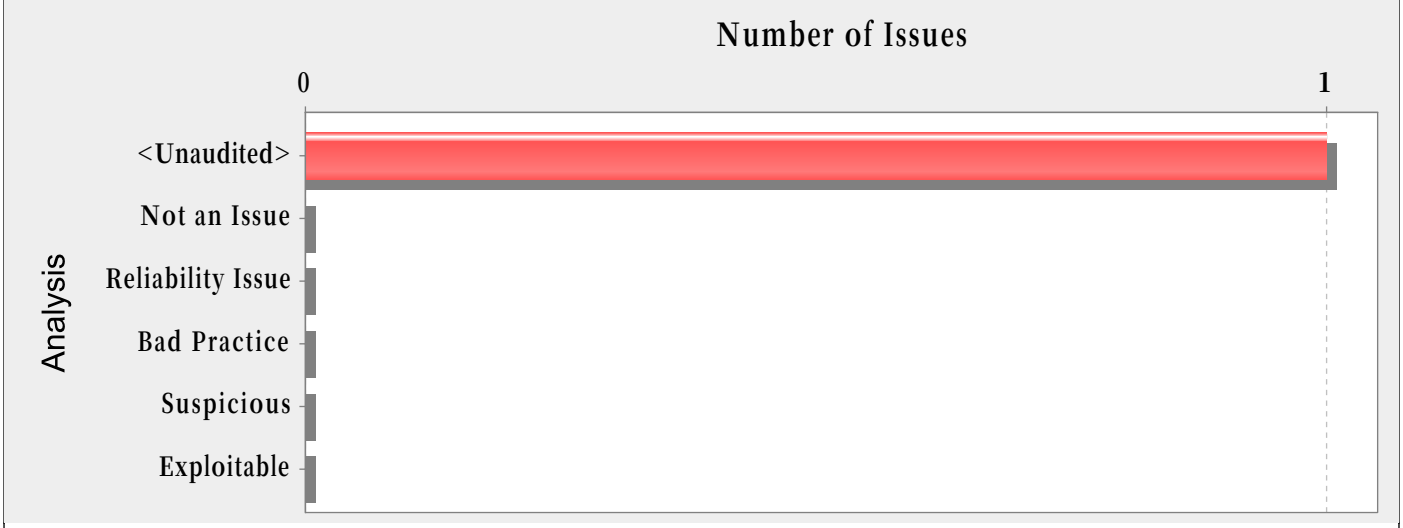
Recommendations:
应尽可能使用 HTTPS 等安全协议与服务器交换数据。

duk_debug.js, line 1847 (Insecure Transport)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	调用 duk_debug.js 中第 1847 行的 get() 会使用未加密的协议（而非加密的协议）与服务器通信。		
Sink:	duk_debug.js:1847 FunctionPointerCall: get()		

```
1845    app.post('/source', this.handleSourcePost.bind(this));
1846    app.post('/sourceList', this.handleSourceListPost.bind(this));
1847    app.get('/heapDump.json', this.handleHeapDumpGet.bind(this));
1848    app.use('/', express.static(__dirname + '/static'));
```

Category: Cross-Site Scripting: Reflected (1 Issues)



Abstract:

文件 x.php 的第 7 行向一个 Web 浏览器发送了未验证的数据，从而导致该浏览器执行恶意代码。

Explanation:

Cross-Site Scripting (XSS) 漏洞在以下情况下发生：

1. 数据通过一个不可信赖的数据源进入 Web 应用程序。对于 Reflected XSS，不可信赖的数据源通常为 Web 请求，而对于 Persisted（也称为 Stored）XSS，该数据源通常为数据库或其他后端数据存储。

2. 未检验包含在动态内容中的数据，便将其传送给了 Web 用户。

传送到 Web 浏览器的恶意内容通常采用 JavaScript 代码片段的形式，但也可能会包含一些 HTML、Flash 或者其他任意一种可以被浏览器执行的代码。基于 XSS 的攻击手段花样百出，几乎是无穷无尽的，但通常它们都会包含传输给攻击者的私有数据（如 Cookie 或者其他会话信息）。在攻击者的控制下，指引受害者进入恶意的网络内容；或者利用易受攻击的站点，对用户的机器进行其他恶意操作。

例 1：下面的 PHP 代码片段可从一个 HTTP 请求中读取雇员 ID eid，并将其显示给用户。

```
<?php
$eid = $_GET['eid'];
...
?>
...
<?php
echo "Employee ID: $eid";
?>
```

如果 eid 只包含标准的字母或数字文本，这个例子中的代码就能正确运行。如果 eid 里有包含元字符或源代码中的值，那么 Web 浏览器就会像显示 HTTP 响应那样执行代码。

起初，这个例子似乎是不会轻易遭受攻击的。毕竟，有谁会输入导致恶意代码在自己电脑上运行的 URL 呢？真正的危险在于攻击者会创建恶意的 URL，然后采用电子邮件或社交工程的欺骗手段诱使受害者访问此 URL 的链接。当受害者单击这个链接时，他们不知不觉地通过易受攻击的网络应用程序，将恶意内容带到了自己的电脑中。这种对易受攻击的 Web 应用程序进行盗取的机制通常被称为反射式 XSS。

例 2：下面的 PHP 代码片段可根据一个给定的雇员 ID 查询数据库，并显式出相应的雇员姓名。

```
<?php...
$con = mysql_connect($server,$user,$password);
...
$result = mysql_query("select * from emp where id="+eid);
$row = mysql_fetch_array($result)
echo 'Employee name: ', mysql_result($row,0,'name');
...
?>
```

如同Example 1，如果对 name 的值处理得当，该代码就能正常地执行各种功能；如若处理不当，就会对代码的漏洞利用行为无能为力。同样，这段代码看起来似乎危险较小，因为 name 的值是从数据库中读取的，而且这些内容明显是由应用程序管理的。然而，如果 name 的值来自用户提供的数据，数据库就会成为恶意内容传播的通道。如果不对数据库中存储的所有数据进行恰当的输入验证，那么攻击者就可以在用户的 Web 浏览器中执行恶意命令。这种类型的漏洞利用称为 Persistent XSS（或 Stored XSS），它极其隐蔽，因为数据存储导致的间接行为会增大辨别威胁的难度，并使多个用户受此攻击影响的可能性提高。XSS 漏洞利用首先会在网站上为访问者提供一个“留言簿”。攻击者会在这些留言簿的条目中嵌入 JavaScript，接下来所有访问该留言簿页面的访问者都会执行这些恶意代码。

正如例子中所显示的，XSS 漏洞是由于 HTTP 响应中包含了未经验证的数据代码而引起的。受害者遭受 XSS 攻击的途径有三种：

- 如Example 1 中所示，系统从 HTTP 请求中直接读取数据，并在 HTTP 响应中返回数据。当攻击者诱使用户为易受攻击的 Web 应用程序提供危险内容，而这些危险内容随后会反馈给用户并在 Web 浏览器中执行时，就会发生 Reflected XSS 漏洞利用。发送恶意内容最常用的方法是，将恶意内容作为一个参数包含在公开发布或通过电子邮件直接发送给受害者的 URL 中。以这种手段构造的 URL 已成为多种网络钓鱼阴谋的核心，攻击者会借此诱骗受害者访问指向易受攻击站点的 URL。当该站点将攻击者的内容反馈给受害者后，便会执行这些内容，接下来会将用户计算机中的各种私密信息（比如可能包含会话信息的 Cookie）传输给攻击者，或者执行其他恶意活动。

- 如Example 2 中所示，应用程序将危险数据存储在数据库或其他可信赖的数据存储中。这些危险数据随后会被读回到应用程序中，并包含在动态内容中。在以下情况下会发生 Persistent XSS 漏洞利用：攻击者将危险内容注入到数据存储中，而这些危险内容随后会被读取并包含在动态内容中。从攻击者的角度看，注入恶意内容的最佳位置莫过于显示给许多用户或显示给特定相关用户的区域。这些相关用户通常在应用程序中具备较高的特权，或者可以与敏感数据交互，这些数据对攻击者来说具有利用价值。如果某一个用户执行了恶意内容，攻击者就有可能以该用户的名义执行某些需要特权的操作，或者获得该用户个人敏感数据的访问权。

— 应用程序之外的数据源将危险数据储存在一个数据库或其他数据存储器中，随后这些危险数据被当作可信赖的数据回写到应用程序中，并储存在动态内容中。

Recommendations:

针对 XSS 的解决方法是，确保在适当位置进行验证，并检验其属性是否正确。

由于 XSS 漏洞出现在应用程序的输出中包含恶意数据时，因此，合乎逻辑的做法是在数据流出应用程序的前一刻对其进行验证。然而，由于 Web 应用程序常常会包含复杂而难以理解的代码，用以生成动态内容，因此，这一方法容易产生遗漏错误（遗漏验证）。降低这一风险的有效途径是对 XSS 也执行输入验证。

由于 Web 应用程序必须验证输入信息以避免其他漏洞（如 SQL Injection），因此，一种相对简单的解决方法是，加强一个应用程序现有的输入验证机制，将 XSS 检测包括其中。尽管有一定的价值，但 XSS 输入验证并不能取代严格的输出验证。应用程序可能通过共享的数据存储或其他可信赖的数据源接受输入，而该数据存储所接受的输入源可能并未执行适当的输入验证。因此，应用程序不能间接地依赖于该数据或其他任意数据的安全性。这就意味着，避免 XSS 漏洞的最佳方法是验证所有进入应用程序以及由应用程序传送至用户端的数据。

针对 XSS 漏洞进行验证最安全的方式是，创建一份安全字符白名单，允许其中的字符出现在 HTTP 内容中，并且只接受完全由这些经认可的字符组成的输入。例如，有效的用户名可能仅包含字母数字字符，电话号码可能仅包含 0-9 的数字。然而，这种解决方法在 Web 应用程序中通常是行不通的，因为许多字符对浏览器来说都具有特殊的含义，在写入代码时，这些字符仍应被视为合法的输入，比如一个 Web 设计版就必须接受带有 HTML 代码片段的输入。

更灵活的解决方法称为黑名单方法，但其安全性较差，这种方法在进行输入之前就有选择地拒绝或避免了潜在的危险字符。为了创建这样一个列表，首先需要了解对于 Web 浏览器具有特殊含义的字符集。虽然 HTML 标准定义了哪些字符具有特殊含义，但是许多 Web 浏览器会设法更正 HTML 中的常见错误，并可能在特定的上下文中认为其他字符具有特殊含义，这就是我们不鼓励使用黑名单作为阻止 XSS 的方法的原因。卡耐基梅隆大学 (Carnegie Mellon University) 软件工程学院 (Software Engineering Institute) 下属的 CERT(R) (CERT(R) Coordination Center) 合作中心提供了有关各种上下文中认定的特殊字符的具体信息 [1]：

在有关块级别元素的内容中（位于一段文本的中间）：

— "<" 是一个特殊字符，因为它可以引入一个标签。

— "&" 是一个特殊字符，因为它可以引入一个字符实体。

— ">" 是一个特殊字符，之所以某些浏览器将其认定为特殊字符，是基于一种假设，即该页的作者本想在前面添加一个 "<"，却错误地将其遗漏了。

下面的这些原则适用于属性值：

— 对于外加双引号的属性值，双引号是特殊字符，因为它们标记了该属性值的结束。

— 对于外加单引号的属性值，单引号是特殊字符，因为它们标记了该属性值的结束。

— 对于不带任何引号的属性值，空格字符（如空格符和制表符）是特殊字符。

- "&" 与某些特定变量一起使用时是特殊字符，因为它引入了一个字符实体。

例如，在 URL 中，搜索引擎可能会在结果页面内提供一个链接，用户可以点击该链接来重新运行搜索。可以将这一方法运用于编写 URL 中的搜索查询语句，这将引入更多特殊字符：

— 空格符、制表符和换行符是特殊字符，因为它们标记了 URL 的结束。

- "&" 是特殊字符，因为它可引入一个字符实体或分隔 CGI 参数。

- 非 ASCII 字符（即 ISO-8859-1 编码表中所有大于 127 的字符）不允许出现在 URL 中，因此这些字符在此环境下被视为特殊字符。

— 在服务器端对在 HTTP 转义序列中编码的参数进行解码时，必须过滤掉输入中的 "%" 符号。例如，当输入中出现 "%68%65%6C%6C%6F" 时，只有从输入的内容中过滤掉 "%"，上述字符串才能在网页上显示为 "hello"。

在 <SCRIPT> </SCRIPT> 的正文内：

- 如果可以将文本直接插入到已有的脚本标签中，应该过滤掉分号、省略号、中括号和换行符。

服务器端脚本：

— 如果服务器端脚本会将输入中的感叹号 (!) 转换成输出中的双引号 (")，则可能需要对此进行更多过滤。

其他可能出现的情况：

— 如果攻击者在 UTF-7 中提交了一个请求，那么特殊字符 "<" 可能会显示为 "+ADw-"，并可能会绕过过滤。如果输出包含在没有确切定义编码格式的网页中，有些浏览器就会设法根据内容自动识别编码（此处采用 UTF-7 格式）。

在应用程序中确定针对 XSS 攻击执行验证的正确要点，以及验证过程中要考虑的特殊字符之后，下一个难点就是确定验证过程中处理各种特殊字符的方式。如果应用程序认定某些特殊字符为无效输入，那么您可以拒绝任何带有这些无效特殊字符的输入。第二种选择就是采用过滤手段来删除这些特殊字符。然而，过滤的负面作用在于，过滤内容的显示将发生改变。在需要完整显示输入内容的情况下，过滤的这种负面作用可能是无法接受的。

如果必须接受带有特殊字符的输入，并将其准确地显示出来，验证机制一定要对所有特殊字符进行编码，以便删除其具有的含义。官方的 HTML 规范 [2] 提供了特殊字符对应的 ISO 8859-1 编码值的完整列表。

许多应用程序服务器都试图避免应用程序出现 Cross-Site Scripting 漏洞，具体做法是为负责设置特定 HTTP 响应内容的函数提供各种实现方式，以检验是否存在进行 Cross-Site Scripting 攻击必需的字符。不要依赖运行应用程序的服务器，以此确保该应用程序的安全。开发了某个应用程序后，并不能保证在其生命周期中它会在哪些应用程序服务器中运行。由于标准和已知盗取方式的演变，我们不能保证应用程序服务器也会保持同步。

Tips:

1. Fortify 安全编码规则包将就 SQL Injection 和 Access Control 提出警告：当把不可信赖的数据写入数据库时，数据库将出现问题，并且会将数据库当作不可信赖的数据的来源，这会导致 XSS 漏洞。如果数据库在您的环境中是可信赖的资源，则使用自定义筛选器筛选出包含 DATABASE 污染标志或来自数据库源的数据流问题。尽管如此，对所有从数据库中读取的内容进行验证仍然是较好的做法。

2. 虽然使用 URL 对不可信数据进行编码可以防止许多 XSS 攻击，但部分浏览器（尤其是 Internet Explorer 6 和 7 以及其他浏览器）在将数据传递给 JavaScript 解释器之前，会自动在文档对象模型 (DOM) 中的特定位置对其内容进行解码。为了反映出其危险之处，规则包不再认为 URL 编码例程足以防御 cross-site scripting 攻击。如果对数据值进行 URL 编码并随后输出，Fortify 将会报告存在 Cross-Site Scripting: Poor Validation 漏洞。

3. 由于 PHP 的动态性，可能会在 PHP 库文件中看到大量结果。可以考虑用一个过滤文件从视图中隐藏特定的结果。关于创建过滤文件的说明，请参见《Fortify Static Code Analyzer（Fortify 静态代码分析器）用户指南》中的“ 高级选项 ”。

x.php, line 7 (Cross-Site Scripting: Reflected)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	文件 x.php 的第 7 行向一个 Web 浏览器发送了未验证的数据，从而导致该浏览器执行恶意代码。		

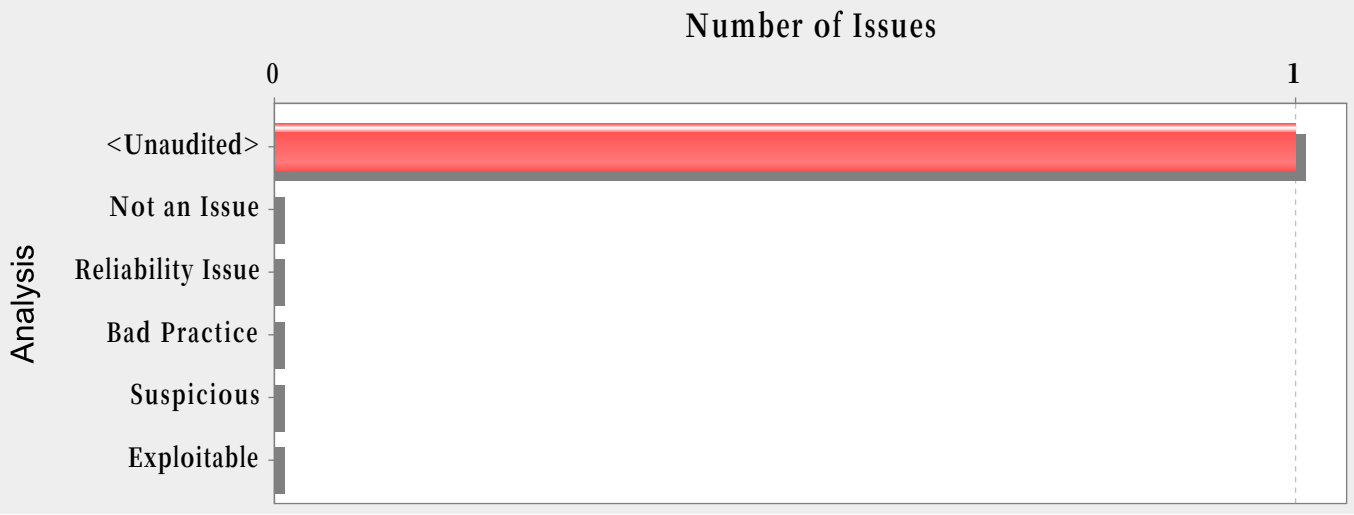
Source: x.php:7 Read $_POST['x']()

```
5  </form>
6
7  <? echo $_POST["x"]; ?>
8
9  </html>
```

Sink: x.php:7 builtin_echo()

```
5  </form>
6
7  <? echo $_POST["x"]; ?>
8
9  </html>
```

Category: Password Management: Insecure Submission (1 Issues)



Abstract:

form.html 中的表单将密码作为第 32 行中 HTTP GET 请求的一部分进行提交，这可能会造成密码被显示、记录或存储在浏览器缓存中。

Explanation:

按照惯例，通常不会将与 HTTP GET 请求有关的参数视为敏感数据，因此 web 服务器会记录这些数据，并将其存储在代理缓存中，同时，web 浏览器也不会刻意隐藏这些数据。作为 HTTP GET 请求的一部分传送密码或其他敏感数据，可能会引起数据的误用，且有可能会被攻击者获取。

示例 1：在以下示例中，新用户密码是通过 HTTP GET 请求提交的。

```
<form method="get">
Name of new user: <input type="text" name="username">
Password for new user: <input type="password" name="user_passwd">
<input type="submit" name="action" value="Create User">
</form>
```

另请注意，method 属性的默认值为 GET，因此忽略该属性会引发相同后果。

Recommendations:

请避免通过 HTTP GET 请求发送敏感数据，例如密码。应使用 HTTP POST，而不是 HTTP GET 将敏感数据由浏览器传送至服务器。

示例 2：在以下示例中，新用户密码是通过 HTTP POST 请求提交的。

```
<form method="post">
Name of new user: <input type="text" name="username">
Password for new user: <input type="password" name="user_passwd">
<input type="submit" name="action" value="Create User">
</form>
```

HTML5 新增了一项功能，即指定 formmethod 属性作为 submit 和 image 输入标签一部分的功能，并且该属性值会覆盖相应表单标签的 method 属性值。

示例 3：在以下示例中，新用户密码也是通过 HTTP POST 请求提交的，这一点是使用 submit 输入标签的 formmethod 属性来指定的。

```
<form method="get">
Name of new user: <input type="text" name="username">
Password for new user: <input type="password" name="user_passwd">
<input type="submit" name="action" value="Create User" formmethod="post">
</form>
```

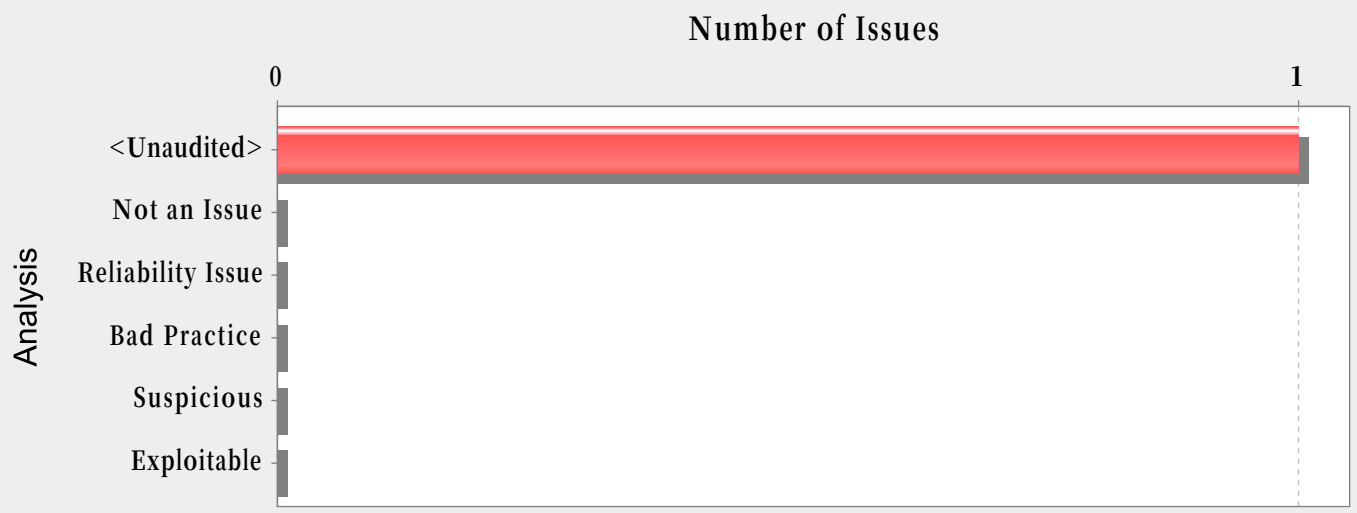
但请注意，如果将 formmethod 属性的值设为 get，无论为相应表单标签的 method 属性指定何值，都将通过 HTTP GET 请求提交该表单。

请避免通过 HTTP 重定向发送敏感数据，因为这样会导致用户的 web 浏览器发出 HTTP GET 请求。应用程序应该将敏感数据存储到会话对象中，从而无需传送回浏览器，或者干脆丢弃这些数据，并要求用户在下次使用时重新输入。对于其他情况，如将敏感数据嵌入到可以自动发布数据的网页中，也存在潜在的危险性，这是因为网页有可能通过代理或 web 浏览器滞留在缓存中。正如我们通常遇到情况，这个实例优先考虑了安全性，而非可用性。

form.html, line 32 (Password Management: Insecure Submission)			
Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	form.html 中的表单将密码作为第 32 行中 HTTP GET 请求的一部分进行提交，这可能会造成密码被显示、记录或存储在浏览器缓存中。		
Sink:	form.html:32 null()		

```
30      <legend>Text inputs:</legend>
31      A text: <input type="text" name="textin"><br />
32      A password: <input type="password" name="passwordin"><br />
33    </fieldset>
```

Category: Privacy Violation: Autocomplete (1 Issues)



Abstract:
form.html 中的表单在第 32 行使用了自动完成功能，该功能允许某些浏览器在历史记录中保留敏感信息。

Explanation:
启用自动完成功能后，某些浏览器会保留会话中的用户输入，以便随后使用该计算机的用户查看之前提交的信息。

Recommendations:
对于表单或敏感输入，显式禁用自动完成功能。通过禁用自动完成功能，之前输入的信息不会在用户输入时以明文形式显示。这也会禁用大多数主要浏览器的“记住密码”功能。

例 1：在 HTML 表单中，通过在 form 标签上将 autocomplete 属性的值显式设置为 off，禁用所有输入字段的自动完成功能。

```
<form method="post" autocomplete="off">
Address: <input name="address" />
Password: <input name="password" type="password" />
</form>
```

例 2：或者，通过在相应的标签上将 autocomplete 属性的值显式设置为 off，禁用特定输入字段的自动完成功能。

```
<form method="post">
Address: <input name="address" />
Password: <input name="password" type="password" autocomplete="off"/>
</form>
```

请注意，autocomplete 属性的默认值为 on。因此，处理敏感输入时请不要忽略该属性。

form.html, line 32 (Privacy Violation: Autocomplete)

Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		
Abstract:	form.html 中的表单在第 32 行使用了自动完成功能，该功能允许某些浏览器在历史记录中保留敏感信息。		

Sink: form.html:32 null()

```
30    <legend>Text inputs:</legend>
31    A text: <input type="text" name="textin"><br />
32    A password: <input type="password" name="passwordin"><br />
33   </fieldset>
```

Issue Count by Category	
Issues by Category	
Insecure Transport	4
Cross-Site Scripting: Reflected	1
Password Management: Insecure Submission	1
Privacy Violation: Autocomplete	1

Issue Breakdown by Analysis

Issues by Analysis