Fortify Security Report

2024-6-21

ASUS

Executive Summary

Issues Overview

On 2024-6-21, a source code review was performed over the rt-n56u code base. 2,017 files, 47,820 LOC (Executable) were scanned and reviewed for defects that could lead to potential security vulnerabilities. A total of 117 reviewed findings were uncovered during the analysis.

Issues by Fortify Priority Order	
High	68
Critical	49

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: C:/Users/ASUS/Desktop/Gitrepo/rt-n56u
Number of Files: 2017
Lines of Code: 47820
Build Label: <No Build Label>

Scan Information

Scan time: 07:46
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
null.NoekeonVects.main
SevenZip.LzmaAlone.Main
SevenZip.LzmaAlone.main

Environment Variables:
null.null.null
os.null.getenv

File System:
null.null.open
null.file.__init__
null.file.read
null.file.readline
null.file.readlines
java.io.FileInputStream.FileInputStream
java.io.FileInputStream.FileInputStream
os.null.open

Private Information:
null.null.null

Standard Input Stream:
null.null.null
null.null.raw_input

Stream:
SevenZip.LzmaBench$MyInputStream.read
java.io.BufferedInputStream.read
java.io.FilterInputStream.read
java.io.InputStream.read
os.null.read

System Information:
null.null.null
null.null.null
null.null.dir
os.null.getcwd
os.null.getpid
os.null.getuid
os.null.listdir
os.null.uname
sys.null.exc_info

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

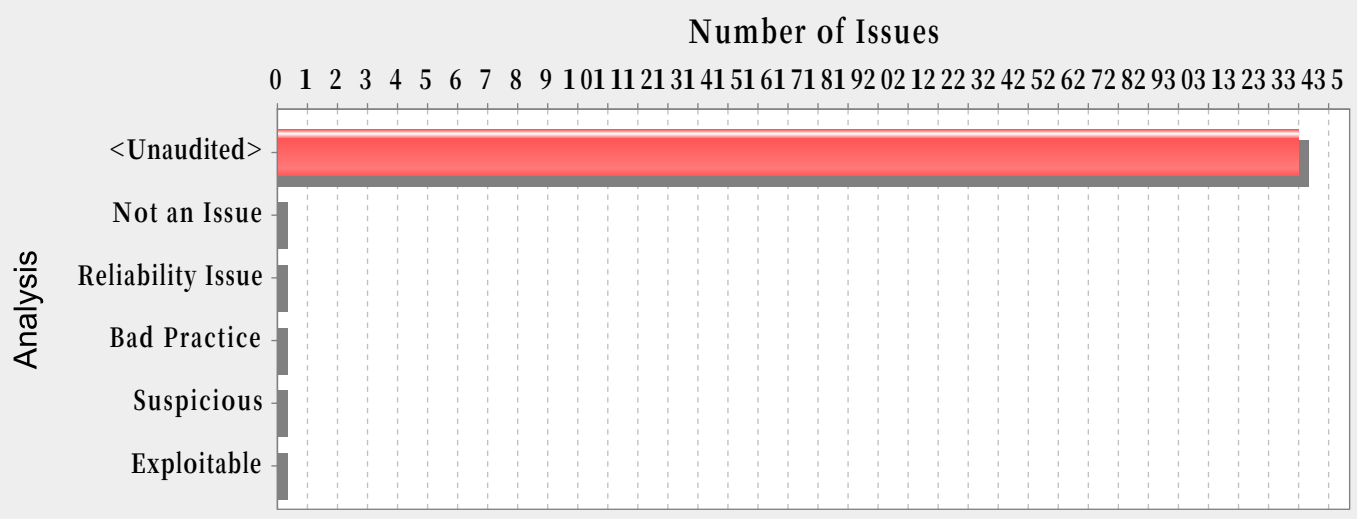
Results Outline

Overall number of results

The scan found 117 issues.

Vulnerability Examples by Category

Category: Password Management: Password in Configuration File (34 Issues)



Abstract:

在配置文件中存储明文密码，可能会危及系统安全。

Explanation:

在配置文件中存储明文密码会使所有能够访问该文件的人都能访问那些用密码保护的资源。程序员有时候认为，他们不可能阻止应用程序被那些能够访问配置文件的攻击者入侵，但是这种想法会导致攻击者发动攻击变得更加容易。健全的 password management 方针从来不会允许以明文形式存储密码。

Recommendations:

绝不能采用明文的形式存储密码。应由管理员在系统启动时输入密码。如果这种方法不切实际，一个安全性较差、但通常都比较恰当的解决办法是将密码模糊化，并把这些去模糊化的资源分散到系统各处，因此，要破译密码，攻击者就必须取得并正确合并多个系统资源。

有些第三方产品宣称可以采用更加安全的方式管理密码。例如，WebSphere Application Server 4.x 用简单的异或加密算法加密数值，但是请不要对诸如此类的加密方式给予完全的信任。WebSphere 以及其他一些应用服务器通常都只提供过期的且相对较弱的加密机制，这对于安全性敏感的环境来说是远远不够的。较为安全的解决方法是由用户自己创建一个新机制，而这也是如今唯一可行的方法。

Tips:

1. Fortify Static Code Analyzer（Fortify 静态代码分析器）会从配置文件中搜索那些用于密码属性的常用名称。当发现密码条目中包含明文时，就会将其标记为问题。

2. 如果配置文件中包含一个默认密码条目，除了需要在配置文件中将其模糊化以外，还需要对其进行修改。

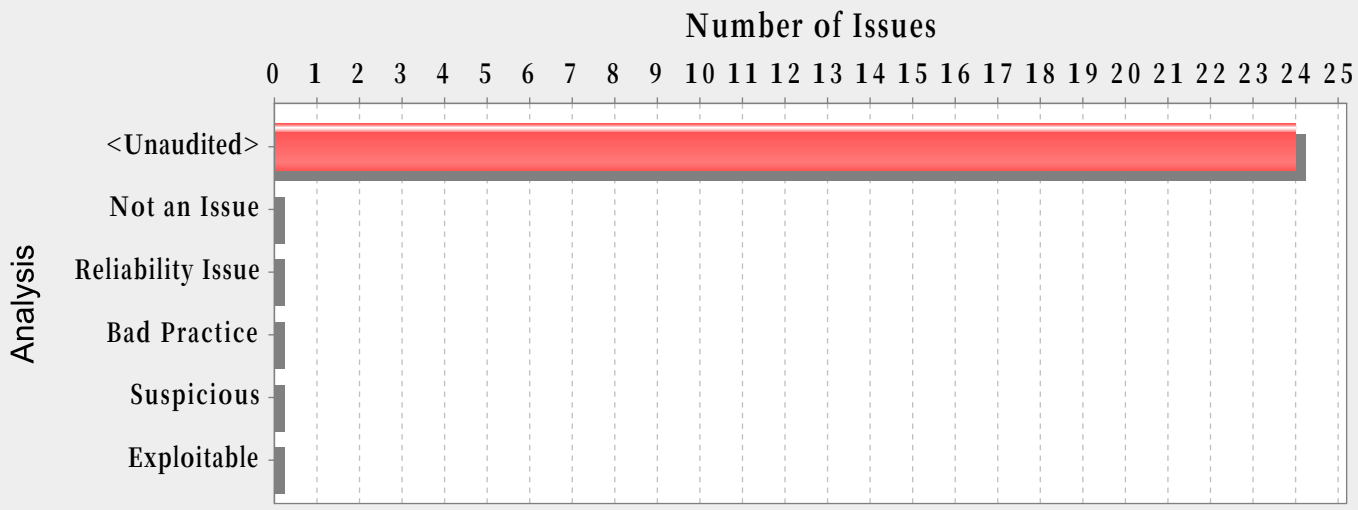
SBE-Appendix2.xml, line 1244 (Password Management: Password in Configuration File)

Fortify Priority:	High	**Folder**	High
Kingdom:	Environment		
Abstract:	在配置文件中存储明文密码，可能会危及系统安全。		

Sink: SBE-Appendix2.xml:1244 null()

```
1242
1243        <para>
1244        Is it necessary to specify <smbconfoption name="encrypt passwords">Yes</smbconfoption>
1245        when Samba-3 is configured as a domain member?
1246        </para>
```

Category: Privacy Violation: Autocomplete (24 Issues)



Abstract:
Advanced_DDNS_Content.asp 中的表单在第 531 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。

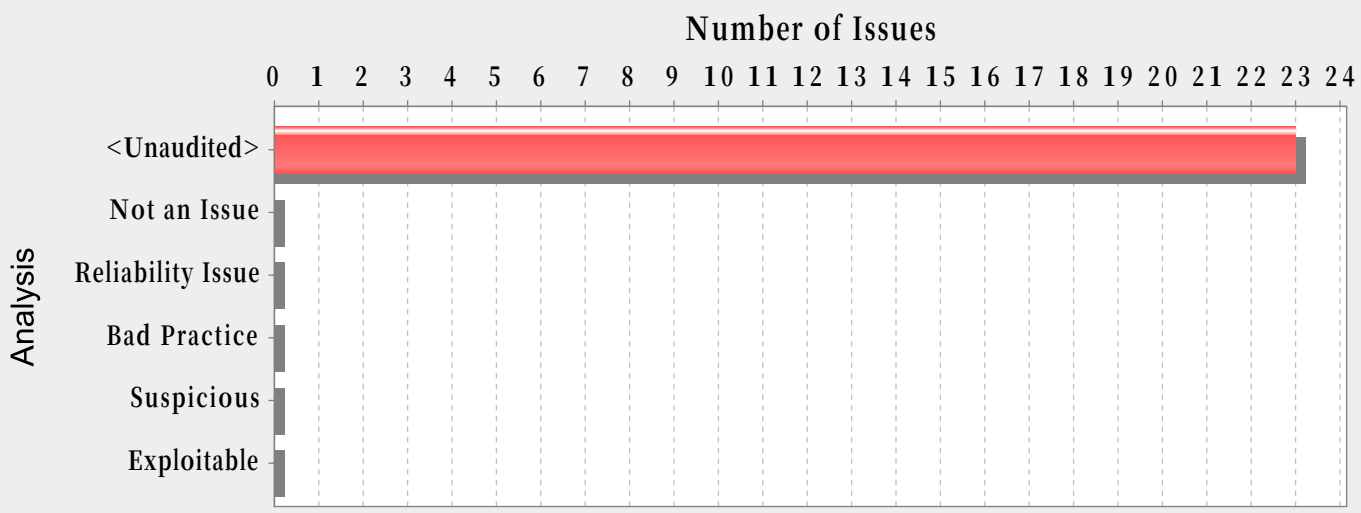
Advanced_DDNS_Content.asp, line 531 (Privacy Violation: Autocomplete)

Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		
Abstract:	Advanced_DDNS_Content.asp 中的表单在第 531 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: Advanced_DDNS_Content.asp:531 null()

```
529                                 <td>
530                                   <div class="input-append">
531                                     <input type="password" maxlength="64" class="input" size="32" name="ddns_passwd_x"
id="ddns_passwd_x" style="width: 175px;" value="<% nvram_get_x("","ddns_passwd_x"); %>" />
532                                     <button style="margin-left: -5px;" class="btn" type="button"
onclick="passwordShowHide('ddns_passwd_x')"><i class="icon-eye-close"></i></button>
533                                   </div>
```

Category: Key Management: Hardcoded Encryption Key (23 Issues)



Abstract:

Hardcoded 加密密钥可能会削弱系统安全性，一旦出现安全问题将无法轻易修正。

Explanation:

使用硬编码方式处理加密密钥绝非好方法。这不仅是因为所有项目开发人员都可以使用通过硬编码方式处理的加密密钥，而且还会使解决这一问题变得极其困难。在代码投入使用之后，必须对软件进行修补才能更改加密密钥。如果受加密密钥保护的帐户遭受入侵，系统所有者将必须在安全性和可用性之间做出选择。

示例：下列代码使用 hardcoded 加密密钥来加密信息：

```
...
from Crypto.Ciphers import AES
encryption_key = b'_hardcoded__key_'
cipher = AES.new(encryption_key, AES.MODE_CFB, iv)
msg = iv + cipher.encrypt(b'Attack at dawn')
...
```

此代码将成功运行，但任何有权访问此代码的人都可以获得加密密钥。一旦程序发布，除非修补该程序，否则可能无法更改硬编码的加密密钥 _hardcoded__key_。心怀不轨的雇员可以利用其对此信息的访问权限来破坏系统加密的数据。

Recommendations:

绝不能对加密密钥进行硬编码。通常情况下，应对加密密钥加以模糊化，并在外部资源文件中进行管理。如果在系统中采用明文的形式存储加密密钥，任何有足够权限的人即可读取加密密钥，还可能误用这些密码。

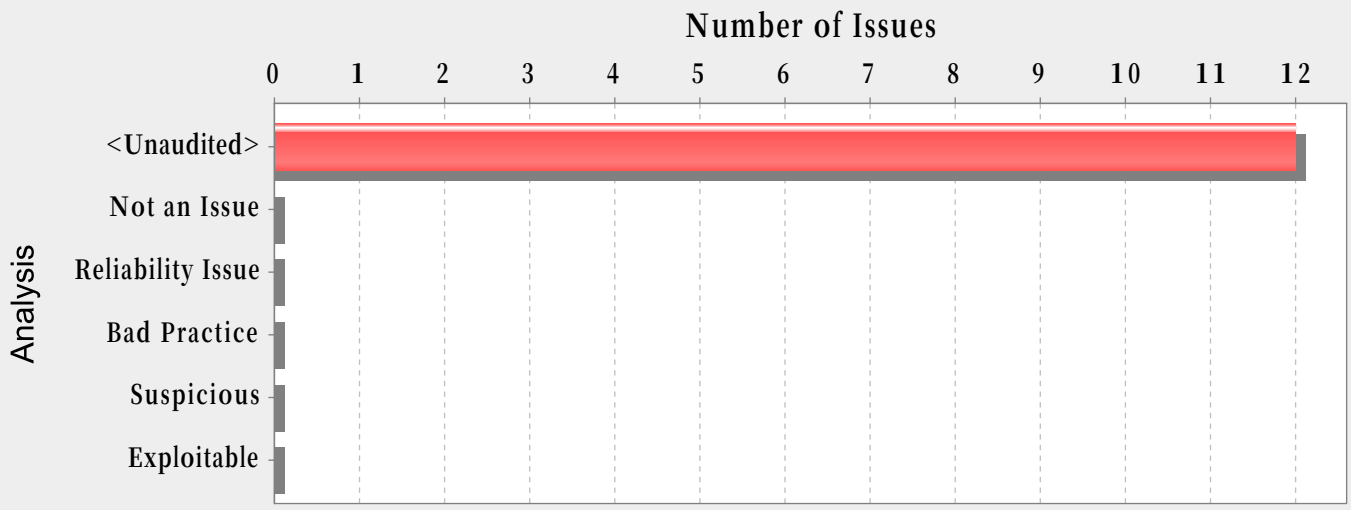
demo_dynamic.py, line 285 (Key Management: Hardcoded Encryption Key)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	Hardcoded 加密密钥可能会削弱系统安全性，一旦出现安全问题将无法轻易修正。		

Sink: demo_dynamic.py:285 VariableAccess: key()

```
283  if SHOW_CHACHA_EXAMPLE:
284    print('-'*60)
285    key    = b'hownowbrowncow\x00\x00' # exactly 16 or 32 bytes
286    rounds = 12                # common values: 8, 12, 20
287    iv     = b'123456789012'       # exactly 12 bytes
```

Category: Password Management: Password in HTML Form (12 Issues)



Abstract:
对 HTML 表单中的密码字段进行填充可能会危及系统安全。

Explanation:
对 HTML 表单中的密码字段进行填充会让任何人都能在 HTML 源中看到它们的值。此外，存储在密码字段中的敏感信息可能会被代理或浏览器缓存。

Recommendations:
请不要填充密码类型的表单字段。

示例：在 HTML 表单中，请不要设置敏感输入的 value 属性。

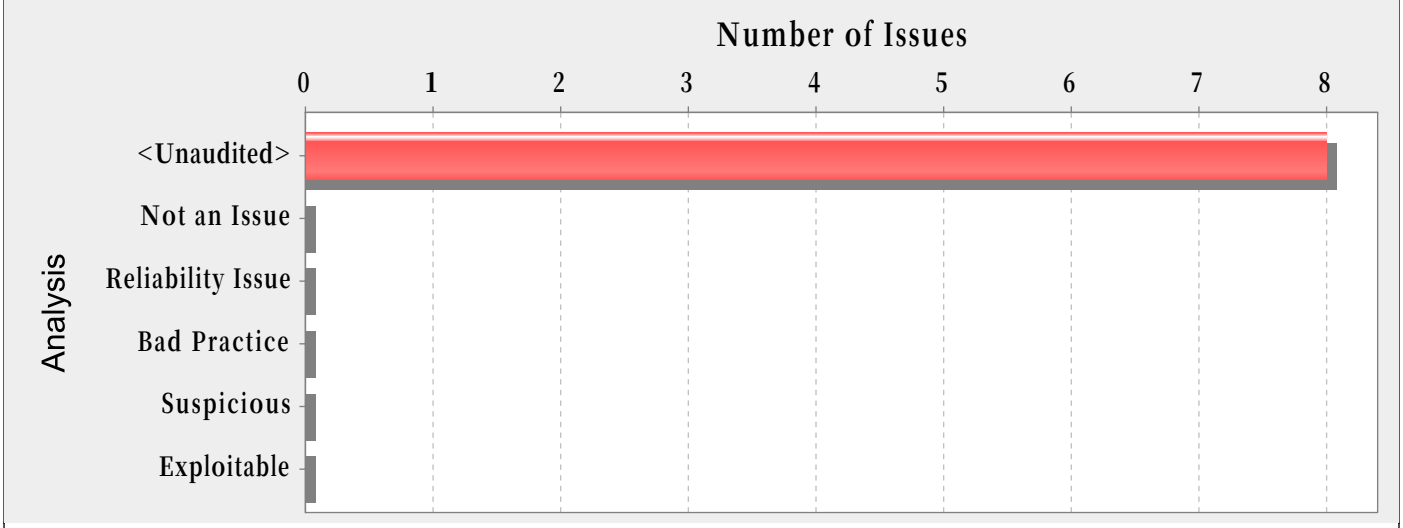
```
<input type="password" />
```

Advanced_DDNS_Content.asp, line 531 (Password Management: Password in HTML Form)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	对 HTML 表单中的密码字段进行填充可能会危及系统安全。		
Sink:	Advanced_DDNS_Content.asp:531 null()		

```
529                         <td>
530                           <div class="input-append">
531                             <input type="password" maxlength="64" class="input" size="32" name="ddns_passwd_x"
id="ddns_passwd_x" style="width: 175px;" value="<% nvram_get_x("","ddns_passwd_x"); %>" />
532                             <button style="margin-left: -5px;" class="btn" type="button"
onclick="passwordShowHide('ddns_passwd_x')"><i class="icon-eye-close"></i></button>
533                           </div>
```

Category: Weak Encryption (8 Issues)



Abstract:
调用 Advanced_Wireless2g_Content.asp 中第 226 行的 change_key_des() 会使用弱加密算法，无法保证敏感数据的保密性。

Explanation:
过时的加密算法（如 DES）无法再为敏感数据的使用提供足够的保护。加密算法依赖于密钥大小，这是确保加密强度的主要方法之一。加密强度通常以生成有效密钥所需的时间和计算能力来衡量。计算能力的提高使得在合理的时间内获得较小的加密密钥成为可能。例如，在二十世纪七十年代首次开发出 DES 算法时，要破解在该算法中使用的 56 位密钥将面临巨大的计算障碍，但今天，使用常用设备能在不到一天的时间内破解 DES。

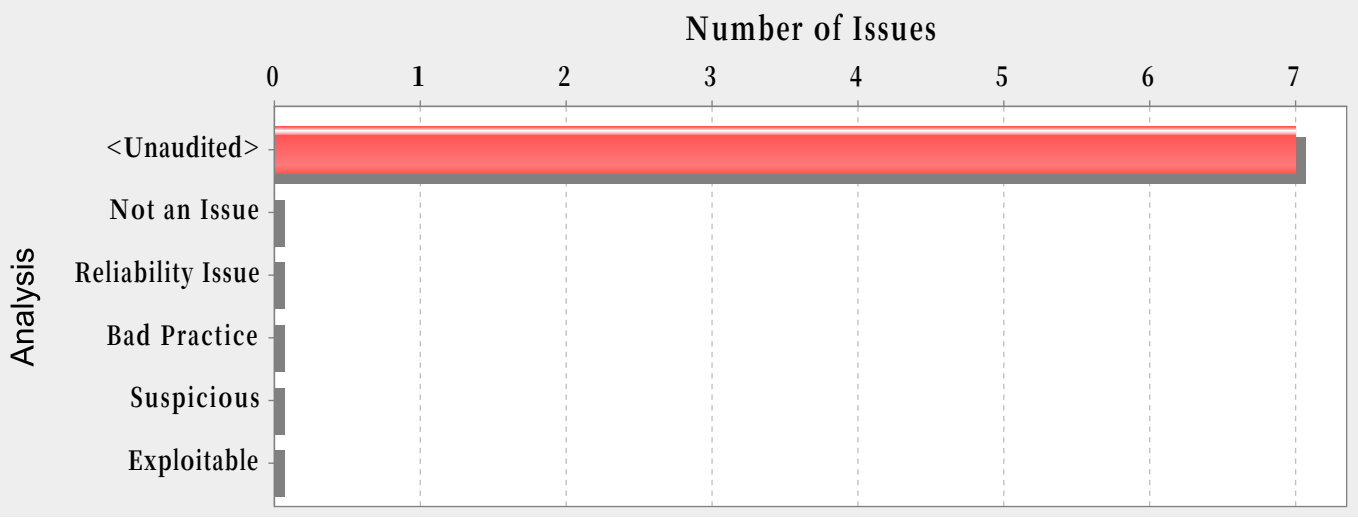
Recommendations:
使用密钥较大的强加密算法来保护敏感数据。作为 DES 的备选强加密算法的示例包括 Rijndael（高级加密标准，简称 AES）和 Triple DES (3DES)。在选择一种算法之前，应首先确定您的组织是否对某个特定算法和实施实现了标准化。

Advanced_Wireless2g_Content.asp, line 226 (Weak Encryption)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	调用 Advanced_Wireless2g_Content.asp 中第 226 行的 change_key_des() 会使用弱加密算法，无法保证敏感数据的保密性。		
Sink:	Advanced_Wireless2g_Content.asp:226 FunctionCall: change_key_des()		

```
224  }
225
226  function change_key_des(){
227      var objs = getElementsByName_iefix("span", "key_des");
228      var wep_type = document.form.rt_wep_x.value;
```

Category: Password Management: Empty Password (7 Issues)



Abstract:

Empty password 可能会危及系统安全，并且无法轻易修正出现的安全问题。

Explanation:

使用空密码绝非好方法。一旦代码投入使用，解决这一问题将变得极其困难。除非对软件进行修补，否则将无法更改密码。如果受空密码保护的帐户遭受入侵，系统所有者将必须在安全性和可用性之间做出选择。

示例：以下代码使用 empty password 来连接应用程序和检索地址簿条目：

```
...
obj = new XMLHttpRequest();
obj.open('GET','/fetchusers.jsp?id='+form.id.value,'true','scott','');
...
```

这些代码将成功运行，但任何人员在知道用户名后均可进行访问。

Recommendations:

密码始终不能为空。一般来说，应对密码加以模糊化，并在外部资源中进行管理。如果将密码以明文形式存储在网站中任意位置，会造成任何有充分权限的人读取和无意中误用密码。对于需要输入密码的 JavaScript 引用，最好在连接时就提示用户输入密码。

Tips:

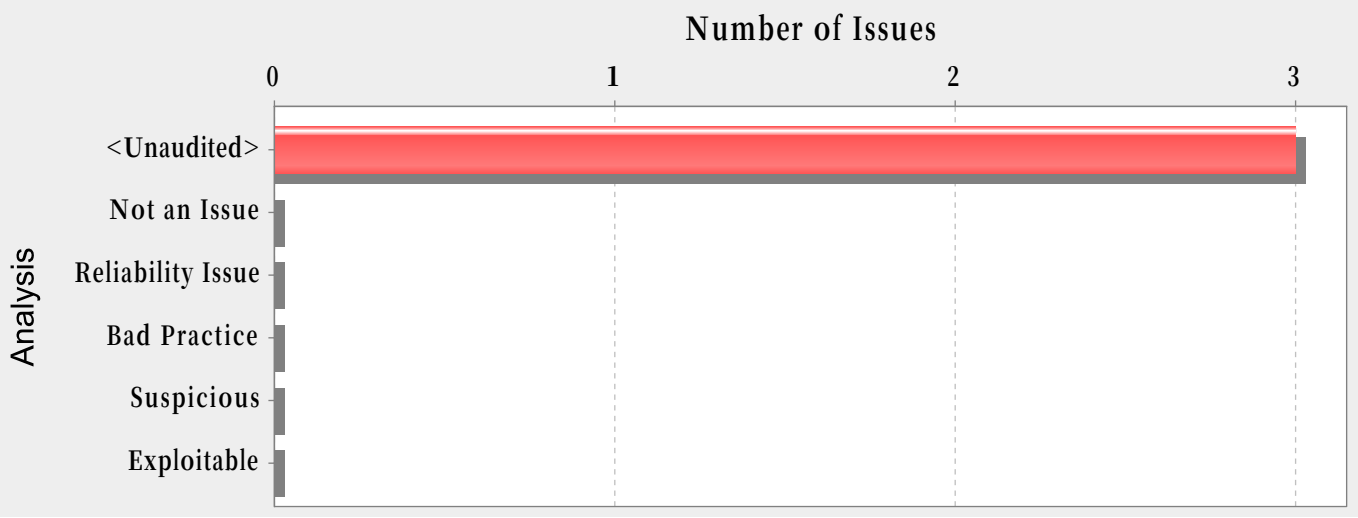
1. 避免在源代码中使用 empty password，还要避免使用默认密码。如果 empty password 处于缺省状态，则需要修改密码，使其不出现在源代码中。

2. 在识别 null 密码、空密码和硬编码密码时，默认规则只会考虑包含 password 一词的字段和变量。但是，使用 Fortify Custom Rules Editor（Fortify 自定义规则编辑器）提供的“ Password Management（密码管理）”向导可轻松创建用于在自定义命名字段和变量中检测 password management 问题的规则。

transmission.min.js, line 1 (Password Management: Empty Password)

Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		
Abstract:	Empty password 可能会危及系统安全，并且无法轻易修正出现的安全问题。		
Sink:	transmission.min.js:1 FieldAccess: password()		
1	var transmission={SessionId:"",isInitialized:!1,host:"",port:"9091",path:"/transmission/rpc",rpcpath:"../rpc",fullpath:"",on:{torrentCountChange:null,postError:null},username:"",password:"",_status:{stopped:0,checkwait:1,check:2,downloadwait:3,download:4,seedwait:5,seed:6,actively:101},_trackerStatus:{inactive:0,waiting:1,queued:2,active:3},options:{getFolders:!0,getTarckers:!0},headers:{},trackers:{},islocal:!1,downloadDirs:new Array,getSessionId:function(t,e){var s={type:"POST",url:this.fullpa...		

Category: Privacy Violation (3 Issues)



Abstract:

stf.py 文件会错误地处理第 101 行的机密信息，从而危及到用户的个人隐私，这是一种非法行为。

Explanation:

Privacy Violation 会在以下情况下发生：

1. 用户私人信息进入了程序。

2. 数据被写到了一个外部介质，例如控制台、file system 或网络。

示例 1：以下代码包含了一个日志记录语句，该语句通过在日志文件中存储记录信息跟踪添加到数据库中的各条记录信息。在存储的其他数值中，有一个是 getPassword() 函数的返回值，该函数会返回与该帐户关联且由用户提供的明文密码。

```
pass = getPassword();
logger.warning('%s: %s %s %s', id, pass, type, tsstamp)
```

Example 1 中的代码会将明文密码记录到应用程序的事件日志中。虽然许多开发人员认为事件日志是存储数据的安全位置，但这不是绝对的，特别是涉及到隐私问题时。

可以通过多种方式将私人数据输入到程序中：

— 以密码或个人信息的形式直接从用户处获取

— 由应用程序访问数据库或者其他数据存储形式

— 间接地从合作者或者第三方处获取

有时，某些数据并没有贴上私人数据标签，但在特定的上下文中也有可能成为私人信息。比如，通常认为学生的学号不是私人信息，因为学号中并没有明确而公开的信息用以定位特定学生的个人信息。但是，如果学校用学生的社会保障号码生成学号，那么这时学号应被视为私人信息。

安全和隐私似乎一直是一对矛盾。从安全的角度看，您应该记录所有重要的操作，以便日后可以鉴定那些非法的操作。然而，当其中牵涉到私人数据时，这种做法就存在一定风险了。

虽然私人数据处理不当的方式多种多样，但常见风险来自于盲目信任。程序员通常会信任运行程序的操作环境，因此认为将私人信息存放在文件系统、注册表或者其他本地控制的资源中是值得信任的。尽管已经限制了某些资源的访问权限，但仍无法保证所有访问这些资源的个体都是值得信任的。例如，2004 年，一个不道德的 AOL 员工将大约 9200 万个私有客户电子邮件地址卖给了一个通过垃圾邮件进行营销的境外赌博网站 [1]。

鉴于此类备受瞩目的信息盗取事件，私人信息的收集与管理正日益规范化。要求各个组织应根据其经营地点、所从事的业务类型及其处理的私人数据性质，遵守下列一个或若干个联邦和州的规定：

- Safe Harbor Privacy Framework [3]

- Gramm-Leach Bliley Act (GLBA) [4]

- Health Insurance Portability and Accountability Act (HIPAA) [5]

- California SB-1386 [6]

尽管制定了这些规范，Privacy Violation 漏洞仍时有发生。

Recommendations:

当安全和隐私的需要发生矛盾时，通常应优先考虑隐私的需要。为满足这一要求，同时又保证信息安全的需要，应在退出程序前清除所有私人信息。

为加强隐私信息的管理，应不断改进保护内部隐私的原则，并严格地加以执行。这一原则应具体说明应用程序应该如何处理各种私人数据。在贵组织受到联邦或者州法律的制约时，应确保您的隐私保护原则尽量与这些法律法规保持一致。即使没有针对贵组织的相应法规，您也应当保护好客户的私人信息，以免失去客户的信任。

保护私人数据的最好做法就是最大程度地减少私人数据的暴露。不应允许应用程序、流程处理以及员工访问任何私人数据，除非是出于职责以内的工作需要。正如最小授权原则一样，不应该授予访问者超出其需求的权限，对私人数据的访问权限应严格限制在尽可能小的范围内。

Tips:

1. 要彻底审计所有的 Privacy Violation 漏洞，措施之一就是确保自定义的规则可以识别所有进入程序的私人或敏感信息。无法自动识别多数私人数据信息。若不使用自定义规则，您执行的 Privacy Violation 漏洞检查可能是不完整的。

stf.py, line 101 (Privacy Violation)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	stf.py 文件会错误地处理第 101 行的机密信息，从而危及到用户的个人隐私，这是一种非法行为。		

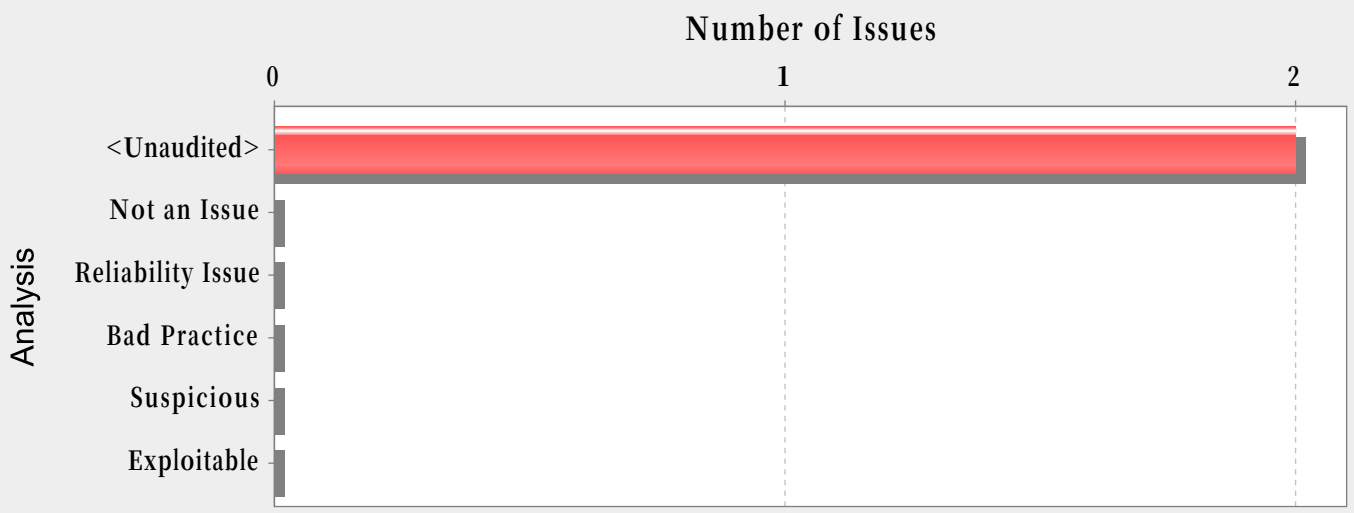
Source: stf.py:35 Read password()

```
33                      "administrator": {"username": username,
34                              "domain": domain,
35                              "password" : password}})
36
37      return server_list
```

Sink: stf.py:101 print()

```
99
100 if __name__ == "__main__":
101     print get_server(platform = "nt")
```

Category: Key Management: Empty Encryption Key (2 Issues)



Abstract:
空加密密钥可能会削弱安全性，一旦出现安全问题将无法轻易修正。

Explanation:
使用空加密密钥绝非好方法。这不仅是因为使用空加密密钥会大幅减弱由良好的加密算法提供的保护，而且还会使解决这一问题变得极其困难。在问题代码投入使用之后，除非对软件进行修补，否则将无法更改空加密密钥。如果受空加密密钥保护的帐户遭受入侵，系统所有者将必须在安全性和可用性之间做出选择。

示例：以下代码会将加密密钥变量初始化为空字符串。

```
...
from Crypto.Ciphers import AES
cipher = AES.new("", AES.MODE_CFB, iv)
msg = iv + cipher.encrypt(b'Attack at dawn')
...
```

不仅任何可以访问此代码的人可以确定它使用的是空加密密钥，而且任何掌握最基本破解技术的人都更有可能成功解密所有加密数据。一旦程序发布，要更改空加密密钥，就必须进行软件修补。雇员可以利用手中掌握的信息访问权限入侵系统。即使攻击者只能访问应用程序的可执行文件，他们也可以提取使用了空加密密钥的证据。

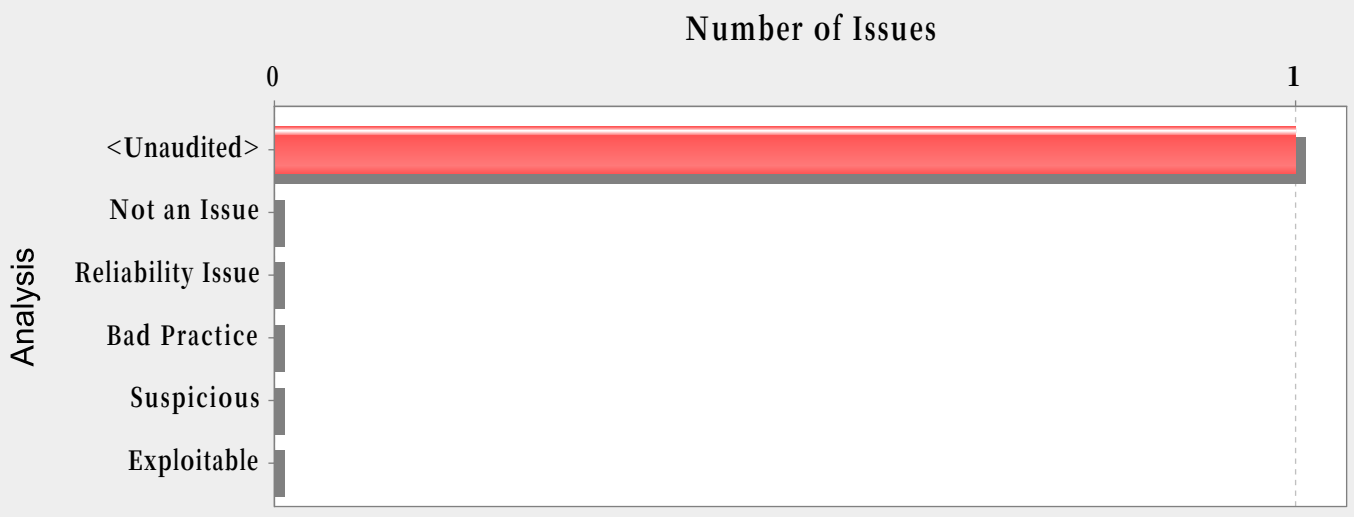
Recommendations:
加密密钥绝不能为空。通常情况下，应对加密密钥加以模糊化，并在外部资源文件中进行管理。如果在系统中采用明文的形式存储加密密钥（空或非空），任何有足够权限的人即可读取加密密钥，还可能误用这些加密密钥。

mountstats.py, line 370 (Key Management: Empty Encryption Key)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	空加密密钥可能会削弱安全性，一旦出现安全问题将无法轻易修正。		
Sink:	mountstats.py:370 VariableAccess: key()		

```
368   """
369   ms_dict = dict()
370   key = ''
371
372   f = file(filename)
```

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



Abstract:

jquery.form.js 中的方法 fileUploadXhr() 向第 346 行的 Web 浏览器发送非法数据，从而导致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。对于基于 DOM 的 XSS，将从 URL 参数或浏览器中的其他值读取数据，并使用客户端代码将其重新写入该页面。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。对于基于 DOM 的 XSS，任何时候当受害人的浏览器解析 HTML 页面时，恶意内容都将作为 DOM（文档对象模型）创建的一部分执行。

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 JavaScript 代码片段可从 URL 中读取雇员 ID eid，并将其显示给用户。

```
<SCRIPT>
var pos=document.URL.indexOf("eid=")+4;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
```

示例 2：考虑使用 HTML 表单：

```
<div id="myDiv">
Employee ID: <input type="text" id="eid"><br>
...
<button>Show results</button>
</div>
<div id="resultsDiv">
...
</div>
```

下面的 jQuery 代码片段可从表单中读取雇员 ID，并将其显示给用户。

```
$(document).ready(function(){
$("#myDiv").on("click", "button", function(){
var eid = $("#eid").val();
$("resultsDiv").append(eid);
...
});
});
```

如果文本输入中 ID 为 eid 的雇员 ID 仅包含标准字母数字文本，则这些代码示例可正确运行。如果 eid 中的某个值包含元字符或源代码，则 Web 浏览器就会在显示 HTTP 响应时执行该代码。

示例 3：以下代码显示了 React 应用程序中基于 DOM 的 XSS 示例：

```
let element = JSON.parse(getUntrustedInput());
ReactDOM.render(<App>
{element}
</App>);
```

在Example 3 中，如果攻击者可以控制从 getUntrustedInput() 检索到的整个 JSON 对象，他们可能就能够使 React 将 element 呈现为一个组件，从而可以使用他们自己控制的值传递具有 dangerouslySetInnerHTML 的对象，这是一种典型的 Cross-Site Scripting 攻击。

最初，这些代码看起来似乎不会轻易遭受攻击。毕竟，有谁会输入包含可在自己电脑上运行的恶意代码的内容呢？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。

正如例子中所显示的，XSS 漏洞是由于 HTTP 响应中包含了未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：

- 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。

— 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器中。这些危险数据随后会被回写到应用程序中，并包含在动态内容中。Persistent XSS 盗取发生在如下情况：攻击者将危险内容注入到数据存储器中，且该存储器之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一个面向许多用户，尤其是相关用户显示的区域。相关用户通常在应用程序中具备较高的特权，或相互之间交换敏感数据，这些数据对攻击者来说有利用价值。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所有的敏感数据的访问权限。

— 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. 较早版本的 React 更容易受到 Cross-Site Scripting 攻击，因为它会控制整个文档。而较新的版本则会使用 Symbols 标识 React 组件，从而可以防止漏洞利用，但不支持 Symbol（本机或通过 Polyfill）的较早浏览器（例如所有版本的 Internet Explorer）仍然容易受到攻击。其他类型的 Cross-Site Scripting 攻击适用于所有浏览器以及各个版本的 React。

jquery.form.js, line 346 (Cross-Site Scripting: DOM)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	jquery.form.js 中的方法 fileUploadXhr() 向第 346 行的 Web 浏览器发送非法数据，从而导致浏览器执行恶意代码。		

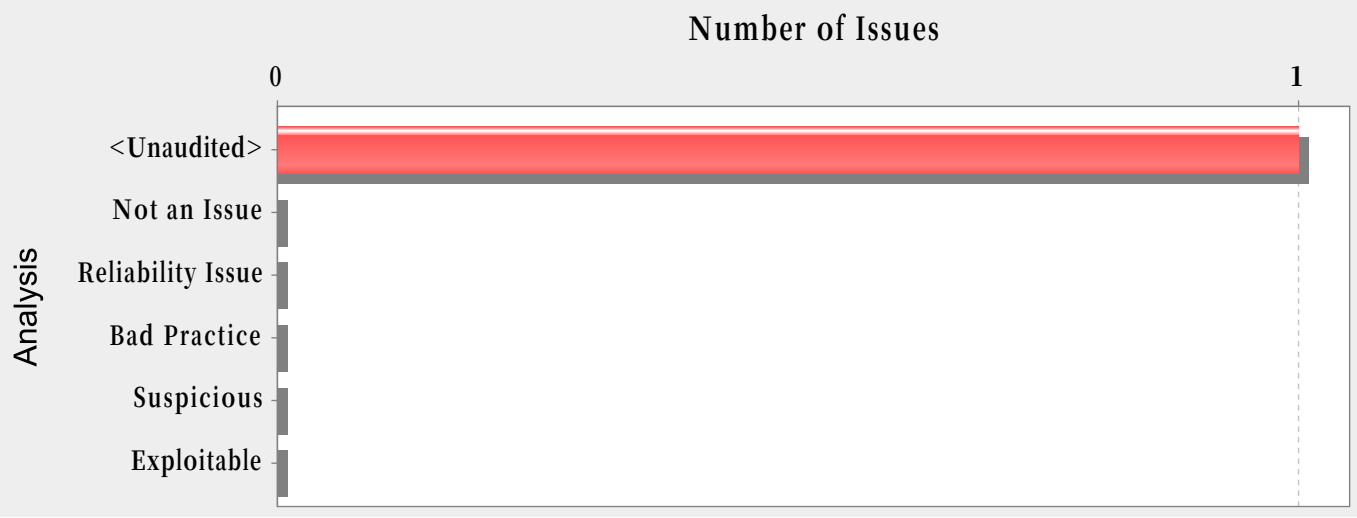
Source: jquery.form.js:115 Read window.location()

```
113
114     url = (typeof action === 'string') ? $.trim(action) : '';
115     url = url || window.location.href || '';
116     if (url) {
117       // clean url (don't include hash vaue)
```

Sink: jquery.form.js:346 jQuery~~wrapper.ajax()

```
344         }
345       };
346       return $.ajax(s);
347     }
```

Category: Dynamic Code Evaluation: Code Injection (1 Issues)



Abstract:

prototype.js 文件将未验证的用户输入解析为第 706 行的源代码。在运行时中解析用户控制的指令，会让攻击者有机会执行恶意代码。

Explanation:

许多现代编程语言都允许动态解析源代码指令。这使得程序员可以执行基于用户输入的动态指令。当程序员错误地认为由用户直接提供的指令仅会执行一些无害的操作时（如对当前的用户对象进行简单的计算或修改用户的状态），就会出现 code injection 漏洞：然而，若不经过适当的验证，用户指定的操作可能并不是程序员最初所期望的。

示例：在这一典型的代码注入示例中，应用程序实施的基本计算器允许用户指定要执行的命令。

```
...
userOp = form.operation.value;
calcResult = eval(userOp);
...
```

如果 operation 参数的值为良性值，程序就可以正常运行。例如，当该值为 "8 + 7 * 2" 时，calcResult 变量被赋予的值将为 22。然而，如果攻击者指定的语言操作既有可能是有效的，又有可能是恶意的，那么，只有在对主进程具有完全权限的情况下才能执行这些操作。如果底层语言提供了访问系统资源的途径或允许执行系统命令，这种攻击甚至会更加危险。对于 JavaScript，攻击者还可以利用这种漏洞进行 cross-site scripting 攻击。

Recommendations:

在任何时候，都应尽可能地避免动态的代码解析。如果程序的功能要求对代码进行动态的解析，您可以通过以下方式将此种攻击的可能性降低到最小：尽可能的限制程序中动态执行的代码数量，将此类代码应用到特定的应用程序和上下文中的基本编程语言的子集。

如果需要执行动态代码，应用程序绝不应当直接执行和解析未验证的用户输入。而应采用间接方法：创建一份合法操作和数据对象列表，用户可以指定其中的内容，并且只能从中进行选择。利用这种方法，就绝不会直接执行由用户提供的输入。

prototype.js, line 706 (Dynamic Code Evaluation: Code Injection)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	prototype.js 文件将未验证的用户输入解析为第 706 行的源代码。在运行时中解析用户控制的指令，会让攻击者有机会执行恶意代码。		

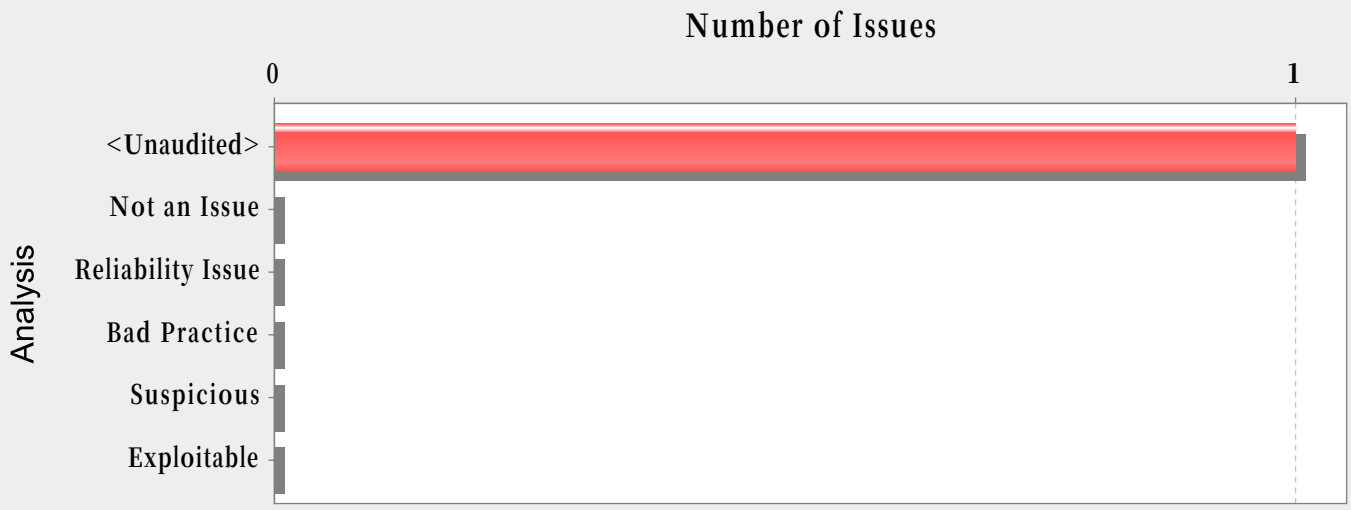
Source: prototype.js:706 Read responseText()

```
704  evalResponse: function() {
705    try {
706      return eval(this.transport.responseText);
707    } catch (e) {
708      this.dispatchException(e);
```

Sink: prototype.js:706 environment~object.eval()

```
704  evalResponse: function() {
705    try {
706      return eval(this.transport.responseText);
707    } catch (e) {
708      this.dispatchException(e);
```

Category: Password Management: Hardcoded Password (1 Issues)



Abstract:

Hardcoded password 可能会危及系统安全性，并且无法轻易修正出现的安全问题。

Explanation:

使用硬编码方式处理密码绝非好方法。这不仅是因为所有项目开发人员都可以使用通过硬编码方式处理的密码，而且还会使解决这一问题变得极其困难。在代码投入使用之后，除非对软件进行修补，否则将无法更改密码。如果受密码保护的帐户遭受入侵，系统所有者将必须在安全性和可用性之间做出选择。

示例：以下代码使用 hardcoded password 来连接应用程序和检索地址簿条目：

```
...
obj = new XMLHttpRequest();
obj.open('GET','/fetchusers.jsp?id='+form.id.value,'true','scott','tiger');
...
```

该代码会正常运行，但是任何能够访问其中所包含的网页的人都能得到这个密码。

Recommendations:

绝不能对密码进行硬编码。通常情况下，应对密码加以模糊化，并在外部资源文件中进行管理。如果将密码以明文形式存储在网站中任意位置，会造成任何有充分权限的人读取和无意中误用密码。对于需要输入密码的 JavaScript 引用，最好在连接时就提示用户输入密码。

Tips:

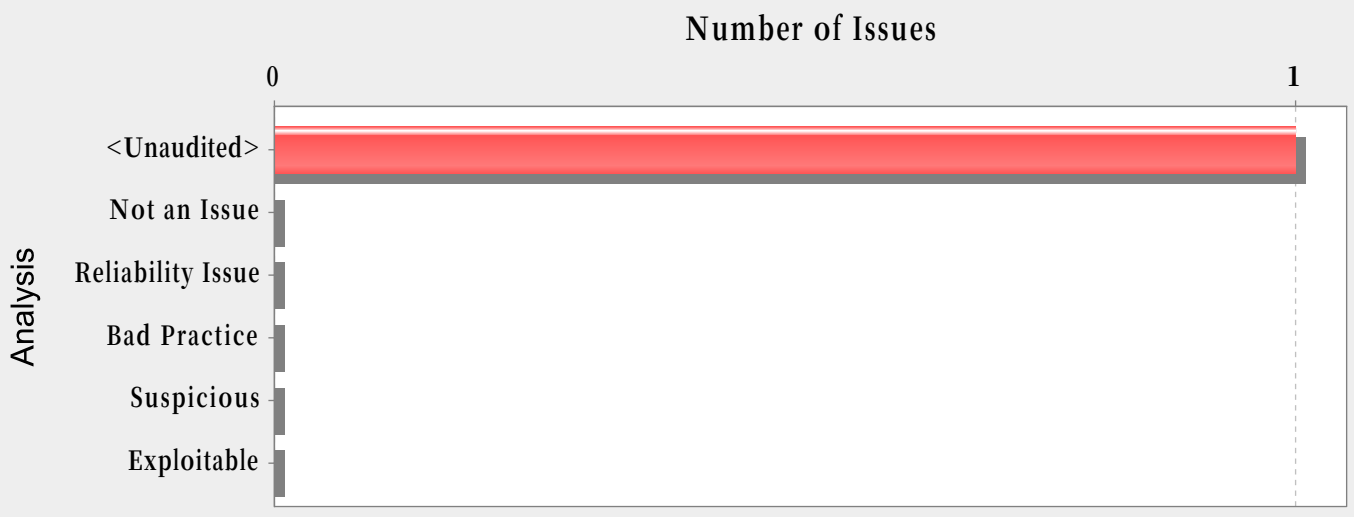
1. 避免在源代码中对密码进行硬编码，还要避免使用默认密码。如果 hardcoded password 处于缺省状态，则需要修改密码，使其不出现在源代码中。

2. 在识别 null 密码、空密码和硬编码密码时，默认规则只会考虑包含 password 一词的字段和变量。但是，使用 Fortify Custom Rules Editor（Fortify 自定义规则编辑器）提供的“ Password Management（密码管理）”向导可轻松创建用于在自定义命名字段和变量中检测 password management 问题的规则。

jquery.easyui.min.js, line 7538 (Password Management: Hardcoded Password)

Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		
Abstract:	Hardcoded password 可能会危及系统安全性，并且无法轻易修正出现的安全问题。		
Sink:	jquery.easyui.min.js:7538 FieldAccess: passwordChar()		
7536	return $.extend({},$.fn.textbox.parseOptions(_56d),$.parser.parseOptions(_56d,["passwordChar",{checkInterval:"number",lastDelay:"number",revealed:"boolean",showEye:"boolean"}]));		
7537	};		
7538	$.fn.passwordbox.defaults=$.extend({},$.fn.textbox.defaults,{passwordChar:"%u25CF",checkInterval:200,lastDelay:500,revealed:false,showEye:true,inputEvents:{focus:_561,blur:_565},val:function(_56e){		
7539	return $(_56e).parent().prev().passwordbox("getValue");		
7540	}});		

Category: Privacy Violation: Shoulder Surfing (1 Issues)



Abstract:
aidisk.asp 中的表单以明文形式将密码写入屏幕上的行 255 中。

Explanation:
密码无需对其所有者可见，且不能对他人可见。如果以明文形式显示密码，附近的任何人都能看到，这会危及系统安全。对于计算机安全性，肩窥指利用直接观察技巧来获取信息，例如从某个人的背后窥视。在拥挤的公共环境中，肩窥更有可能得逞。这种威胁主要是针对适用于公共以及私密环境中的移动设备。

Recommendations:
切勿以明文形式显示密码。用点或星号来模糊显示该字段字符，而不是以明文显示。

示例：在 HTML 表单中，将敏感输入的 type 属性设置为 password。

```
<input name="password" type="password" />
```

请注意，type 属性的默认值为 text，而不是 password。因此，处理敏感输入时请不要忽略该属性。

aidisk.asp, line 255 (Privacy Violation: Shoulder Surfing)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	aidisk.asp 中的表单以明文形式将密码写入屏幕上的行 255 中。		
Sink:	aidisk.asp:255 null()		

```
253   <input type="hidden" name="dummyShareway" id="dummyShareway" value="<% nvram_get_x("", "dummyShareway"); %>">
254   <input type="hidden" name="account" id="account" value="">
255   <input type="hidden" name="password" id="password" value="">
256   <input type="hidden" name="protocol" id="protocol" value="">
257   <input type="hidden" name="mode" id="mode" value="">
```

Issue Count by Category

Issues by Category	
Password Management: Password in Configuration File	34
Privacy Violation: Autocomplete	24
Key Management: Hardcoded Encryption Key	23
Password Management: Password in HTML Form	12
Weak Encryption	8
Password Management: Empty Password	7
Privacy Violation	3
Key Management: Empty Encryption Key	2
Cross-Site Scripting: DOM	1
Dynamic Code Evaluation: Code Injection	1
Password Management: Hardcoded Password	1
Privacy Violation: Shoulder Surfing	1

Issue Breakdown by Analysis

Issues by Analysis