太平保险源代码安全扫描报告

2020-9-23

报表汇总信息

显示测试的基本信息

测试日期:2020-9-23
测试主机名称:WIN-HU802GO6IA1
测试项目代号(BuildID):a3d3ce0084bf4c94802b61e9b78e8549
被测项目文件总数:380
被测项目代码总行数:50816

Issues by Folder

高/中/低风险的漏洞数目

高风险	8
中风险	2
低风险	21

依据风险级别 高/中/低 分别统计

Issues by Category

Refined by: [fortify priority order]:critical

风险级别：高风险

SQL Injection: MyBatis Mapper	6
Password Management: Hardcoded Password	2

Issues by Category

Refined by: [fortify priority order]:high

风险级别：中风险

Password Management: Empty Password	1
Password Management: Hardcoded Password	1

Issues by Category

Refined by: [fortify priority order]:low OR [fortify priority order]:medium

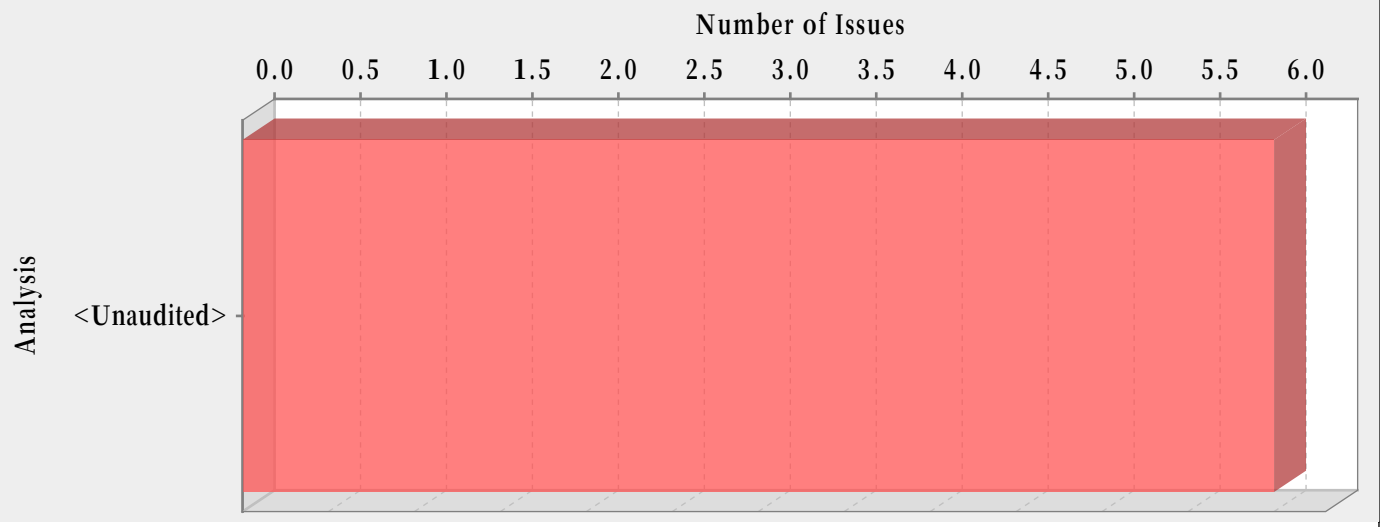
风险级别：低风险

Password Management: Password in Comment	20
Unsafe Reflection	1

测试结果明细

风险级别：高风险

Category: SQL Injection: MyBatis Mapper (6 Issues)



Explanation:

SQL injection 错误在以下情况下发生：

1. 数据从一个不可信赖的数据源进入程序。

2. 数据用于动态地构造一个 SQL 查询。

使用 MyBatis Mapper XML 文件可以指定 SQL 指令中的动态参数，通常使用 # 字符来定义它们，如：

```
<select id="getItems" parameterType="domain.company.MyParamClass" resultType="MyResultMap">
SELECT *
FROM items
WHERE owner = #{userName}
</select>
```

变量名称周围带有括号的 # 字符表示 MyBatis 将使用 userName 变量创建参数化查询。但是，MyBatis 还允许使用 $ 字符将变量直接连接到 SQL 指令，使其易受 SQL injection 攻击。

例 1：以下代码动态地构造并执行了一个 SQL 查询，该查询可以搜索与指定名称相匹配的项。该查询仅会显示条目所有者与被授予权限的当前用户一致的条目。

```
<select id="getItems" parameterType="domain.company.MyParamClass" resultType="MyResultMap">
SELECT *
FROM items
WHERE owner = #{userName}
AND itemname = ${itemName}
</select>
```

但是，由于这个查询是动态构造的，由一个不变的基查询字符串和一个用户输入字符串连接而成，因此只有在 itemName 不包含单引号字符时，才会正确执行这一查询。如果一个用户名为 wiley 的攻击者为 itemName 输入字符串 “ name' OR 'a'='a ”，那么查询就会变成：

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name' OR 'a'='a';
```

附加条件 OR 'a'='a' 会使 WHERE 从句的计算结果始终为 true，因此该查询在逻辑上将等同于一个更为简化的查询：

```
SELECT * FROM items;
```

这种查询的简化会使攻击者绕过查询只应返回经过验证的用户所拥有的条目的要求；而现在的查询则会直接返回所有储存在 items 表中的条目，不论它们的所有者是谁。

例 2：这个例子指出了将不同的恶意数值传递给在例 1 中构造和执行的查询时所带来的各种影响。如果一个用户名为 wiley 在 itemName 中输入字符串“ name'; DELETE FROM items; -- ”，则该查询将会变为以下两个：

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name';

DELETE FROM items;

--'
```

众多数据库服务器，其中包括 Microsoft(R) SQL Server 2000，都可以一次性执行多条用分号分隔的 SQL 指令。对于那些不允许运行用分号分隔的批量指令的数据库服务器，比如 Oracle 和其他数据库服务器，攻击者输入的这个字符串只会导致错误；但是在那些支持这种操作的数据库服务器上，攻击者可能会通过执行多条指令而在数据库上执行任意命令。

注意成对的连字符 (--)；这在大多数数据库服务器上都表示下面的语句将作为注释使用，而不能加以执行 [4]。在这种情况下，注释字符的作用就是删除修改的查询指令中遗留的最后一个单引号。而在那些不允许这样加注注释的数据库中，通常攻击者可以如例 1 那样来攻击。如果攻击者输入字符串“ name'); DELETE FROM items; SELECT * FROM items WHERE 'a'='a ”就会创建如下三个有效指令：

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name';

DELETE FROM items;

SELECT * FROM items WHERE 'a'='a';
```

避免 SQL injection 攻击的传统方法之一是，把它作为一个输入合法性检查的问题来处理，只接受列在白名单中的字符，或者识别并避免那些列在黑名单中的恶意数据。白名单方法是一种非常有效方法，它可以强制执行严格的输入检查规则，但是参数化的 SQL 指令所需维护更少，而且能提供更好的安全保障。而对于通常采用的列黑名单方式，由于总是存在一些小漏洞，所以并不能有效地防止 SQL injection 威胁。例如，攻击者可以：

— 把没有被黑名单引用的值作为目标
— 寻找方法以绕过对某一转义序列元字符的需要
— 使用存储过程来隐藏注入的元字符

手动去除 SQL 查询中的元字符有一定的帮助，但是并不能确保应用程序免受 SQL injection 攻击。

防范 SQL injection 攻击的另外一种常用方式是使用存储过程。虽然存储过程可以阻止某些类型的 SQL injection 攻击，但是对于绝大多数攻击仍无能为力。存储过程有助于避免 SQL injection 的常用方式是限制可作为参数传入的指令类型。但是，有许多方法都可以绕过这一限制，许多危险的表达式仍可以传入存储过程。所以再次强调，存储过程在某些情况下可以避免这种攻击，但是并不能完全保护您的应用系统抵御 SQL injection 的攻击。

Recommendations:

造成 SQL injection 攻击的根本原因在于攻击者可以改变 SQL 查询的上下文，使程序员原本要作为数据解析的数值，被篡改为命令了。当构造一个 SQL 查询时，程序员应当清楚，哪些输入的数据将会成为命令的一部分，而哪些仅仅是作为数据。参数化 SQL 指令可以防止直接窜改上下文，避免几乎所有的 SQL injection 攻击。参数化 SQL 指令是用常规的 SQL 字符串构造的，但是当需要加入用户输入的数据时，它们就需要使用捆绑参数，这些捆绑参数是一些占位符，用来存放随后插入的数据。换言之，捆绑参数可以使程序员清楚地分辨数据库中的数据，即其中有哪些输入可以看作命令的一部分，哪些输入可以看作数据。这样，当程序准备执行某个指令时，它可以详细地告知数据库，每一个捆绑参数所使用的运行时的值，而不会被解析成对该命令的修改。

例 3：可以将例 1 重写为使用参数化 SQL 指令（替代用户输入连续的字符串），如下所示：

```
<select id="getItems" parameterType="domain.company.MyParamClass" resultType="MyResultMap">
SELECT *
FROM items
WHERE owner = #{userName}
AND itemname = #{itemName}
</select>
```

更加复杂的情况常常出现在报表生成代码中，因为这时需要通过用户输入来改变 SQL 指令的命令结构，比如在 WHERE 条件子句中加入动态的约束条件。不要因为这一需求，就无条件地接受连续的用户输入，从而创建查询语句字符串。MyBatis 包含了对在其 XML 架构中构建动态查询进行解释的功能，同时还提供使用参数化查询 [2] 的功能。

例 4：可以将例 3 改写为 XML 中的动态查询，以首先验证 itemName 是否存在，同时仍然使用参数化查询，例如：

```
<select id="getItems" parameterType="domain.company.MyParamClass" resultType="MyResultMap">
SELECT *
FROM items
```

```
WHERE owner = #{userName}
<if test="itemName != null">
AND itemname = #{itemName}
</if>
</select>
```

通过这种方法创建动态查询可以防止由于维持参数化查询的安全而导致的 SQL injection 攻击，从而使其比字符串串联更安全、更易于维护。如果这种方法出于任何原因不可用或不足够，当必须要根据用户输入来改变命令结构时，可以使用间接的方法来防止 SQL injection 攻击：创建一个合法的字符串集合，使其对应于可能要加入到 SQL 指令中的不同元素。在构造一个指令时，可使用来自用户的输入，以便从应用程序控制的值集合中进行选择。

tpIdmAccountInfo-sqlmap.xml, line 104 (SQL Injection: MyBatis Mapper)

Fortify Priority:	Critical	**Folder**	
Kingdom:	Input Validation and Representation		
Abstract:	在 tpIdmAccountInfo-sqlmap.xml 的第 104 行，程序通过可能来自不可信来源的输入构建了 SQL 查询。通过这种调用，攻击者能够修改指令的含义或执行任意 SQL 命令。		

Sink: tpIdmAccountInfo-sqlmap.xml:104 null()

```
102             AND current_page = #{currentPage}
103         </if>
104             <if test="orderStr != null and orderStr != ''">
105         ORDER BY ${orderStr}
106         </if>
```

tpSendSms-sqlmap.xml, line 49 (SQL Injection: MyBatis Mapper)

Fortify Priority:	Critical	**Folder**	
Kingdom:	Input Validation and Representation		
Abstract:	在 tpSendSms-sqlmap.xml 的第 49 行，程序通过可能来自不可信来源的输入构建了 SQL 查询。通过这种调用，攻击者能够修改指令的含义或执行任意 SQL 命令。		

Sink: tpSendSms-sqlmap.xml:49 null()

```
47              AND channel_id = #{channelId}
48          </if>
49              <if test="orderStr != null and orderStr != ''">
50          ORDER BY ${orderStr}
51          </if>
```

tpSysDept-sqlmap.xml, line 311 (SQL Injection: MyBatis Mapper)

Fortify Priority:	Critical	**Folder**	
Kingdom:	Input Validation and Representation		
Abstract:	在 tpSysDept-sqlmap.xml 的第 311 行，程序通过可能来自不可信来源的输入构建了 SQL 查询。通过这种调用，攻击者能够修改指令的含义或执行任意 SQL 命令。		

Sink: tpSysDept-sqlmap.xml:311 null()

```
309             </delete>
310
311         <select id="queryOneTpSysDept" parameterType="java.lang.String"
resultType="com.cntaiping.dataplatform.common.sysdept.model.TpSysDeptVO">
312             SELECT
313                 pk_dept as pkDept ,  code as code ,  inner_code as innerCode ,  par_dep as
parDep ,  s_name as sName ,  n_name as nName ,  update_time as updateTime ,
inner_org_code as innerOrgCode ,  can_type as canType ,  org_type as orgType ,  dept_all as
deptAll ,  old_def as oldDef ,  pk_org as pkOrg ,  create_time as createTime ,  id as id
,  lev as lev ,  ts as ts ,  `order` as `order`  from TP_SYS_DEPT
```

tpSysConf-sqlmap.xml, line 49 (SQL Injection: MyBatis Mapper)

Fortify Priority:	Critical	**Folder**	
Kingdom:	Input Validation and Representation		

Abstract:	在 tpSysConf-sqlmap.xml 的第 49 行，程序通过可能来自不可信来源的输入构建了 SQL 查询。通过这种调用，攻击者能够修改指令的含义或执行任意 SQL 命令。
Sink:	tpSysConf-sqlmap.xml:49 null()

```
47                AND sys_des = #{sysDes}
48              </if>
49                  <if test="orderStr != null and orderStr != ''">
50              ORDER BY ${orderStr}
51              </if>
```

tpSendMail-sqlmap.xml, line 64 (SQL Injection: MyBatis Mapper)

Fortify Priority:	Critical	**Folder**	
Kingdom:	Input Validation and Representation		
Abstract:	在 tpSendMail-sqlmap.xml 的第 64 行，程序通过可能来自不可信来源的输入构建了 SQL 查询。通过这种调用，攻击者能够修改指令的含义或执行任意 SQL 命令。		
Sink:	tpSendMail-sqlmap.xml:64 null()		

```
62                AND channel_id = #{channelId}
63              </if>
64                  <if test="orderStr != null and orderStr != ''">
65              ORDER BY ${orderStr}
66              </if>
```

tpSysDept-sqlmap.xml, line 81 (SQL Injection: MyBatis Mapper)

Fortify Priority:	Critical	**Folder**	
Kingdom:	Input Validation and Representation		
Abstract:	在 tpSysDept-sqlmap.xml 的第 81 行，程序通过可能来自不可信来源的输入构建了 SQL 查询。通过这种调用，攻击者能够修改指令的含义或执行任意 SQL 命令。		
Sink:	tpSysDept-sqlmap.xml:81 null()		

```
79                AND `order` = #{order}
80              </if>
81              <if test="orderStr != null and orderStr != ''">
82              ORDER BY ${orderStr}
83              </if>
```

Category: Password Management: Hardcoded Password (2 Issues)

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。一旦代码投入使用，除非对软件进行修补，否则您再也不能改变密码了。如果帐户中的密码保护减弱，系统所有者将被迫在安全性和可行性之间做出选择。

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. 识别 null password、empty password 和 hardcoded password 时，默认规则只会考虑包含 password 字符的字段和变量。但是，HPE Security Fortify Custom Rules Editor（HPE Security Fortify 自定义规则编辑器）会提供 Password Management 向导，让您轻松创建能够从自定义名称的字段和变量中检测出 password management 问题的规则。

jquery-compat-3.0.0-alpha1.js, line 2070 (Password Management: Hardcoded Password)

Fortify Priority:	Critical	**Folder**	
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-compat-3.0.0-alpha1.js:2070 FieldAccess: password()		

```
2068
2069    // Add button/input type pseudos
2070    for ( i in { radio: true, checkbox: true, file: true, password: true, image: true } ) {
2071        Expr.pseudos[ i ] = createInputPseudo( i );
2072    }
```

jquery-2.1.4.min.js, line 2 (Password Management: Hardcoded Password)

Fortify Priority:	Critical	**Folder**	
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-2.1.4.min.js:2 FieldAccess: password()		

```
0    /*! jQuery v2.1.4 | (c) 2005, 2015 jQuery Foundation, Inc. | jquery.org/license */
```

1	!function(a,b){"object"==typeof module&&"object"==typeof module.exports?module.exports=a.document?b(a,!0):function(a){if(!a.document)throw new Error("jQuery requires a window with a document");return b(a)}:b(a)}("undefined"!=typeof window?window:this,function(a,b){var c=[],d=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,j=h.hasOwnProperty,k={},l=a.document,m="2.1.4",n=function(a,b){return new n.fn.init(a,b)},o=/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,p=/^-ms-/,q=/-([\da-z])/gi,r=function...
2	return M.access(a,b,c)},removeData:function(a,b){M.remove(a,b)},_data:function(a,b,c){return L.access(a,b,c)},_removeData:function(a,b){L.remove(a,b)}}),n.fn.extend({data:function(a,b){var c,d,e,f=this[0],g=f&&f.attributes;if(void 0===a){if(this.length&&(e=M.get(f),1===f.nodeType&&!L.get(f,"hasDataAttrs"))){c=g.length;while(c--)g[c]&&(d=g[c].name,0===d.indexOf("data-")&&(d=n.camelCase(d.slice(5)),P(f,d,e[d])));L.set(f,"hasDataAttrs",!0)}return e}return"object"==typeof a?this.each(function(){M.se...
3	void 0===c?d&&"get"in d&&null!==(e=d.get(a,b))?e:(e=n.find.attr(a,b),null==e?void 0:e):null!==c?d&&"set"in d&&void 0!==(e=d.set(a,c,b))?e:(a.setAttribute(b,c+""),c):void n.removeAttr(a,b))},removeAttr:function(a,b){var c,d,e=0,f=b&&b.match(E);if(f&&1===a.nodeType)while(c=f[e++])d=n.propFix[c]||c,n.expr.match.bool.test(c)&&(a[d]=!1),a.removeAttribute(c)},attrHooks:{type:{set:function(a,b){if(!k.radioValue&&"radio"===b&&n.nodeName(a,"input")){var c=a.value;return a.setAttribute("type",b),c&&(a.val...

风险级别：中风险

Category: Password Management: Empty Password (1 Issues)

Explanation:

为密码变量指定空字符串绝非一个好方法。如果使用 empty password 成功通过其他系统的验证，那么相应帐户的安全性很可能会被减弱，原因是其接受了 empty password。如果在为变量指定一个合法的值之前，empty password 仅仅是一个占位符，那么它将给任何不熟悉代码的人造成困惑，而且还可能导致出现意外控制流路径方面的问题。

例 1： 以下代码尝试使用 empty password 连接数据库。

```
...
DriverManager.getConnection(url, "scott", "");
...
```

如果例 1 中的代码成功执行，则表明数据库用户帐户 “ scott ” 配置了一个 empty password，攻击者可以轻松地猜测到这一点。更危险的是，程序一旦发布，更新帐户以使用非 empty password 时，需要对代码进行更改。

例 2： 以下代码可将密码变量初始化为空字符串，并尝试在存储的值中读取密码，且将其与用户提供的值进行比较。

```
...
String storedPassword = "";
String temp;
if ((temp = readPassword()) != null) {
storedPassword = temp;
}
if(storedPassword.equals(userPassword))
// Access protected resources
...
}
...
```

如果 readPassword() 因数据库错误或其他问题而未能取得存储的密码，攻击者只需向 userPassword 提供一个空字符串，就能轻松绕过密码检查。

在移动世界中，由于设备丢失的几率较高，因此密码管理是一个非常棘手的问题。

例 3：以下代码可将用户名和密码变量初始化为空字符串，如果服务器之前未拒绝这些变量当前提出的请求，代码就可从 Android WebView 存储读取凭证，并使用用户名和密码设置身份验证，从而查看受保护页面。

```
...
webview.setWebViewClient(new WebViewClient() {
public void onReceivedHttpAuthRequest(WebView view,
HttpAuthHandler handler, String host, String realm) {
String username = "";
String password = "";
if (handler.useHttpAuthUsernamePassword()) {
String[] credentials = view.getHttpAuthUsernamePassword(host, realm);
username = credentials[0];
password = credentials[1];
}
handler.proceed(username, password);
}
});
...
```

与例 2 相似，如果 useHttpAuthUsernamePassword() 返回 false，攻击者就可以通过提供 empty password 查看受保护页面。

Recommendations:

始终从加密的外部资源读取存储的密码值，并为密码变量指定有意义的值。确保始终不使用 empty password 或 null password 保护敏感资源。

对于 Android 以及其他任何使用 SQLite 数据库的平台来说，SQLCipher 是一个好选择 -- 对 SQLite 数据库的扩展为数据库文件提供了透明的 256 位 AES 加密。因此，凭证可以存储在加密的数据库中。

例 4：以下代码演示了在将所需的二进制码和存储凭证下载到数据库文件后，将 SQLCipher 集成到 Android 应用程序中的方法。

```
import net.sqlcipher.database.SQLiteDatabase;
...
SQLiteDatabase.loadLibs(this);
File dbFile = getDatabasePath("credentials.db");
dbFile.mkdirs();
dbFile.delete();
SQLiteDatabase db = SQLiteDatabase.openOrCreateDatabase(dbFile, "credentials", null);
db.execSQL("create table credentials(u, p)");
db.execSQL("insert into credentials(u, p) values(?, ?)", new Object[]{username, password});
...
```

请注意，对 android.database.sqlite.SQLiteDatabase 的引用可以使用 net.sqlcipher.database.SQLiteDatabase 代替。

要在 WebView 存储上启用加密，需要使用 sqlcipher.so 库重新编译 WebKit。

Tips:

1. 可使用 Fortify Java Annotations、FortifyPassword 和 FortifyNotPassword 来指示哪些字段和变量代表密码。

2. 识别 null password、empty password 和 hardcoded password 时，默认规则只会考虑包含 password 字符的字段和变量。但是，HPE Security Fortify Custom Rules Editor（HPE Security Fortify 自定义规则编辑器）会提供 Password Management 向导，让您轻松创建能够从自定义名称的字段和变量中检测出 password management 问题的规则。

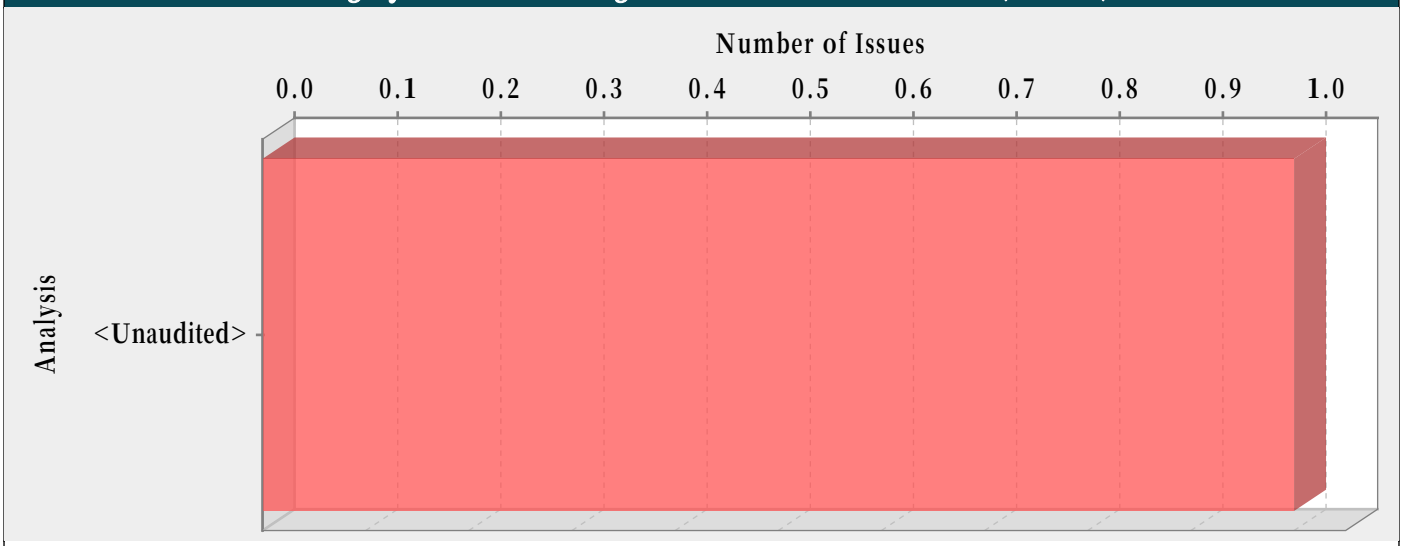
PassWordUtil.java, line 25 (Password Management: Empty Password)

Fortify Priority:	High	**Folder**	
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:	PassWordUtil.java:25 VariableAccess: strEncodePasswd()		

```
23      }
24      String orig_passwd;
```

```
25                  String strEncodePasswd = "";
26                  String strKey;
27                  char code, mid = 0, temp = 0;
```

Category: Password Management: Hardcoded Password (1 Issues)



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。一旦代码投入使用，除非对软件进行修补，否则您再也不能改变密码了。如果帐户中的密码保护减弱，系统所有者将被迫在安全性和可行性之间做出选择。

例 1：以下代码用 hardcoded password 来连接数据库：

```
...
DriverManager.getConnection(url, "scott", "tiger");
...
```

该代码可以正常运行，但是任何有该代码权限的人都能得到这个密码。一旦程序发布，将无法更改数据库用户“ scott ”和密码“ tiger ”，除非是要修补该程序。雇员可以利用手中掌握的信息访问权限入侵系统。更糟的是，如果攻击者能够访问应用程序的字节代码，那么他们就可以利用 javap -c 命令访问已经过反汇编的代码，而在这些代码中恰恰包含着用户使用过的密码值。我们可以从以下看到上述例子的执行结果：

```
javap -c ConnMngr.class
22: ldc   #36; //String jdbc:mysql://ixne.com/rxsql 24: ldc   #38; //String scott 26: ldc   #17; //String tiger
```

在移动世界中，由于设备丢失的几率较高，因此密码管理是一个非常棘手的问题。

例 2：以下代码可使用硬编码的用户名和密码设置身份验证，从而使用 Android WebView 查看受保护页面。

```
...
webview.setWebViewClient(new WebViewClient() {  public void onReceivedHttpAuthRequest(WebView view,
HttpAuthHandler handler, String host, String realm) {    handler.proceed("guest", "allow");  } });
...
```

与例 1 相似，该代码可以正常运行，但是任何有该代码权限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。在系统中采用明文的形式存储密码，会造成任何有充分权限的人读取和无意中误用密码。至少，密码要先经过 hash 处理再存储。

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。一般较为安全的解决方法是采用由用户创建的所有者机制，而这似乎也是目前最好的方法。

对于 Android 以及其他任何使用 SQLite 数据库的平台来说，SQLCipher 是一个好选择 -- 对 SQLite 数据库的扩展为数据库文件提供了透明的 256 位 AES 加密。因此，凭证可以存储在加密的数据库中。

例 3：以下代码演示了在将所需的二进制码和存储凭证下载到数据库文件后，将 SQLCipher 集成到 Android 应用程序中的方法。

```
import net.sqlcipher.database.SQLiteDatabase;
...
SQLiteDatabase.loadLibs(this);  File dbFile = getDatabasePath("credentials.db");  dbFile.mkdirs();  dbFile.delete();  SQLiteDatabase db = SQLiteDatabase.openOrCreateDatabase(dbFile, "credentials", null);  db.execSQL("create table credentials(u, p)");
db.execSQL("insert into credentials(u, p) values(?, ?)", new Object[]{username, password});
```

...

请注意，对 android.database.sqlite.SQLiteDatabase 的引用可以使用 net.sqlcipher.database.SQLiteDatabase 代替。

要在 WebView 存储上启用加密，需要使用 sqlcipher.so 库重新编译 WebKit。

Tips:

1. 可使用 Fortify Java Annotations、FortifyPassword 和 FortifyNotPassword 来指示哪些字段和变量代表密码。

2. 识别 null password、empty password 和 hardcoded password 时，默认规则只会考虑包含 password 字符的字段和变量。但是，HPE Security Fortify Custom Rules Editor（HPE Security Fortify 自定义规则编辑器）会提供 Password Management 向导，让您轻松创建能够从自定义名称的字段和变量中检测出 password management 问题的规则。

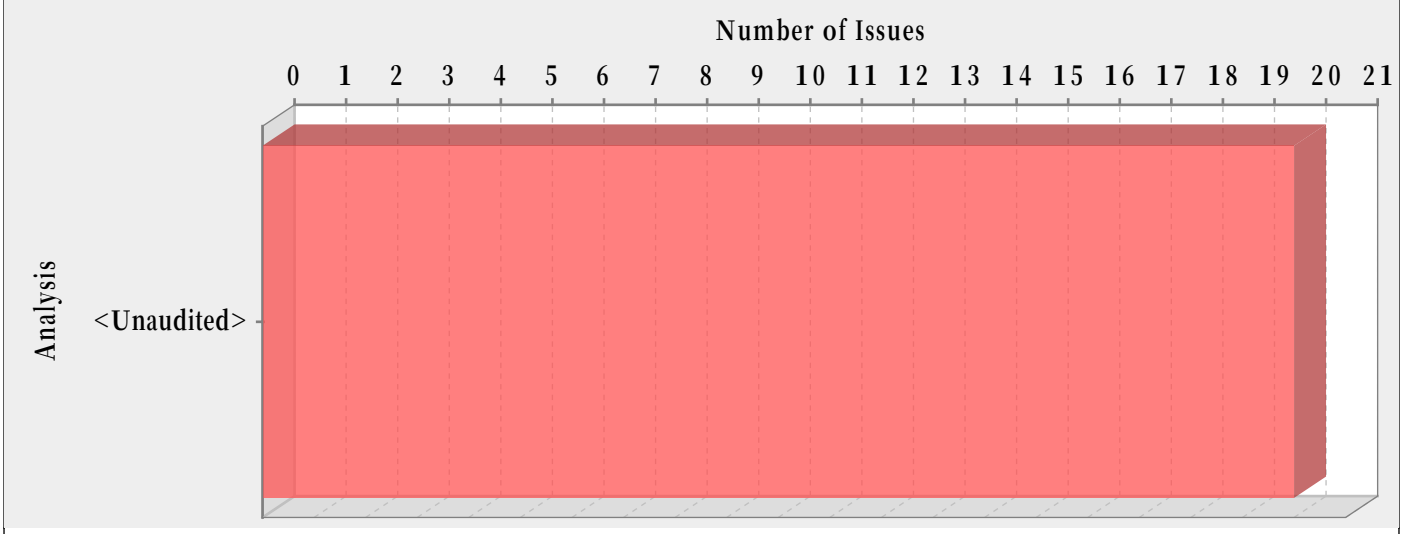
SharedConfiguration.java, line 25 (Password Management: Hardcoded Password)

Fortify Priority:	High **Folder**
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:	SharedConfiguration.java:25 FieldAccess: keystorePassword()

```
23  @JsonIgnore
24  private JKSKeyManager keyManager;
25  private String keystorePassword =******
26  private boolean needsSigning;
27  private String defaultSignatureAlgorithm = SignatureConstants.ALGO_ID_SIGNATURE_RSA_SHA256;
```

风险级别：低风险

Category: Password Management: Password in Comment (20 Issues)



Explanation:

使用硬编码方式处理密码绝非好方法。在注释中存储密码详细信息等同于对密码进行硬编码。这不仅能够使所有项目的开发人员都可以查看密码，而且还会使解决这一问题变得极其困难。一旦代码投入使用，密码便会外泄，除非对软件进行修补，否则您再也不能保护或更改密码了。如果帐户中的密码保护减弱，系统所有者将被迫在安全性和可行性之间做出选择。

示例：以下注释指定连接到数据库的默认密码：

```
...
// Default username for database connection is "scott"  // Default password for database connection is "tiger"
...
```

该代码可以正常运行，但是任何有该代码权限的人都能得到这个密码。一旦程序发布，将无法更改数据库用户“ scott ”和密码“ tiger ”，除非是要修补该程序。雇员可以利用手中掌握的信息访问权限入侵系统。

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。在系统中采用明文的形式存储密码，会造成任何有充分权限的人读取和无意中误用密码。

UserServiceImpl.java, line 35 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	UserServiceImpl.java:35 Comment()		

```
33    private UserMapper userMapper;
34
35    /*@Resource
36    private RoleMapper roleMapper;
```

TpSendSmsService.java, line 125 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	TpSendSmsService.java:125 Comment()		

```
123            }
124            System.out.println("send sms --------->:"+telNo+" content:"+content+" result:"+sendResult);
125            //TPSMSUtils.sendTPSMS(msgs.getUsername(), msgs.getPassword(), msgs);
126
127            Map<String, Object> objMap = BeanUtil.objectToMap(vo);
```

CNTPCryptTest.java, line 13 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	CNTPCryptTest.java:13 Comment()		

```
11  //
12  //      boolean loginResult = CNTPCrypt
13  //              .checkValid("notExsitUser", "", "password",
14  //                      "f936d81f609a01c51509252e948ea90cf12f2f4eaabcccc76d5bc03b6de64bb8");
15  //      Assert.assertFalse(loginResult);
```

UserServiceImpl.java, line 410 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	UserServiceImpl.java:410 Comment()		

```
408     if(pd!=null && !"".equals(pd) && lg!=null && !"".equals(lg)){
409       log.info("鍓嶇    浼犺繃鏉ョ殑瀵嗙爜鍜岀櫥褰曞悕锛屽苟°C伅"+us);
410      // String p = encrypt(user.getPassword(), aesEncryptUtil.getKey(), aesEncryptUtil.getIv()).trim();
411       String p =null;
412       if(us.getPcPwd()!=null){
```

HttpUtil.java, line 117 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	HttpUtil.java:117 Comment()		

```
115   }
116
117   /**
118    * 灏佽    httpclient鐨勮    姹?
119    * @see #jsonHttp(String, String, String, Map)
```

HttpUtil.java, line 57 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		

Sink: HttpUtil.java:57 Comment()

```
55    }
56
57    /**
58     *
59     * @param httpUrl 璇锋眰鐨刄rl 1鎬昏et鏌愪氦鏄撻檺 鍙傛暟?寰ㄧ紡鎷兼帴鏇╋   姹俇rl鍚庨潰 2鏈嶅ost鏌愪氦鏃?姝ゆ?涓鸿   姹傚湴鍧?
```

PassWordUtil.java, line 16 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		

Sink: PassWordUtil.java:16 Comment()

```
14    private static final String M_STRKEY6 = "OP{}|+_)(*&^%$#@!~";
15
16    /**
17     * 鍔犲瘑瀵嗙爜
18     * @param strPasswd
```

HttpRequest.java, line 2396 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		

Sink: HttpRequest.java:2396 Comment()

```
2394    }
2395
2396    /**
2397     * Set the 'Proxy-Authorization' header to given values in Basic auth
2398     * format
```

PassWordUtil.java, line 137 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		

Sink: PassWordUtil.java:137 Comment()

```
135    }
136
137    /*public static void main(String args[]) {
138        PassWord pw = new PassWord();
139        System.out.println(pw.decode(pw.encode("unitele")));
```

jquery-2.1.4.min.js, line 4 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		

Sink: jquery-2.1.4.min.js:4 Comment()

2	!function(a,b){"object"==typeof module&&"object"==typeof module.exports?module.exports=a.document?b(a,!0):function(a){if(!a.document)throw new Error("jQuery requires a window with a document");return b(a)}:b(a)}("undefined"!=typeof window?window:this,function(a,b){var c=[],d=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,j=h.hasOwnProperty,k={},l=a.document,m="2.1.4",n=function(a,b){return new n.fn.init(a,b)},o=/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,p=/^-ms-/,q=/-([\da-z])/gi,r=function...
3	return M.access(a,b,c)},removeData:function(a,b){M.remove(a,b)},_data:function(a,b,c){return L.access(a,b,c)},_removeData:function(a,b){L.remove(a,b)}}),n.fn.extend({data:function(a,b){var c,d,e,f=this[0],g=f&&f.attributes;if(void 0===a){if(this.length&&(e=M.get(f),1===f.nodeType&&!L.get(f,"hasDataAttrs"))){c=g.length;while(c--)g[c]&&(d=g[c].name,0===d.indexOf("data-")&&(d=n.camelCase(d.slice(5)),P(f,d,e[d])));L.set(f,"hasDataAttrs",!0)}return e}return"object"==typeof a?this.each(function(){M.se...
4	void 0===c?d&&"get"in d&&null!==(e=d.get(a,b))?e:(e=n.find.attr(a,b),null==e?void 0:e):null!==c?d&&"set"in d&&void 0!==(e=d.set(a,c,b))?e:(a.setAttribute(b,c+""),c):void n.removeAttr(a,b))},removeAttr:function(a,b){var c,d,e=0,f=b&&b.match(E);if(f&&1===a.nodeType)while(c=f[e++])d=n.propFix[c]||c,n.expr.match.bool.test(c)&&(a[d]=!1),a.removeAttribute(c)},attrHooks:{type:{set:function(a,b){if(!k.radioValue&&"radio"===b&&n.nodeName(a,"input")){var c=a.value;return a.setAttribute("type",b),c&&(a.val...

UserServiceImpl.java, line 162 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		

Sink: UserServiceImpl.java:162 Comment()

```
160   @Override
161   public int resetUserPwd(User user) throws Exception {
162     /* String pwd = encrypt(user.getPassword(), aesEncryptUtil.getKey(), aesEncryptUtil.getIv()).trim();
163     user.setPassword(pwd);
164     User ur=userMapper.selectUserInfoByLoginName(user.getLoginName());
```

HttpUtil.java, line 44 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		

Sink: HttpUtil.java:44 Comment()

```
42    private static Logger logger = LoggerFactory.getLogger(HttpUtil.class);
43
44    /**
45     * 灏佽   httpclient鐨勫   姹?
46     * @param httpUrl 璇锋眰鐨刄rl 1銆佸et鏂规硶鍒欏皢 鍙傛暟缁?褰㈠紡鎷兼帴鍦╱    姹俇rl鍚庨潰 2銆乸ost鏂规硶鍒欏皢鍙傛暟鐢?cmd 鐨勭被鍨嬩紶鍏ワ紝濡?
ゆ?涓鸿    姹傚弬鏁?
```

jquery-compat-3.0.0-alpha1.js, line 8943 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		

Sink: jquery-compat-3.0.0-alpha1.js:8943 Comment()

```
8941        async: true,
8942        contentType: "application/x-www-form-urlencoded; charset=UTF-8",
8943        /*
8944        timeout: 0,
8945        data: null,
```

CNTPCryptTest.java, line 29 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		

Sink: CNTPCryptTest.java:29 Comment()

```
27    //    public static void main(String[] args) {
28    //        boolean loginResult = CNTPCrypt
29    //                .checkValid("notExsitUser", "", "password",
```

```
30                              //                          "f936d81f609a01c51509252e948ea90cf12f2f4eaabcccc76d5bc03b6de64bb8");
31                              //        System.out.println(loginResult);
```

PassWordUtil.java, line 36 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	PassWordUtil.java:36 Comment()		

```
34          }
35          if (includeChineseChar(strPasswd)) {
36  //          orig_passwd = "123456";
37          }
38          strKey = M_STRKEY1 + M_STRKEY2 + M_STRKEY3 + M_STRKEY4 + M_STRKEY5 + M_STRKEY6;
```

WebServiceImpl.java, line 109 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	WebServiceImpl.java:109 Comment()		

```
107             CNTPCrypt sm3 = new CNTPCrypt();
108
109             /*try {
110                 String contentNew = sm3.getEncryptedContent(account.getTpId() +
configService.selectConfigByKey("sys.user.initPassword"));
111                 //鏈　渣璐ヽ彿鐗诲紵璁よ瘉
```

UserServiceImpl.java, line 337 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	UserServiceImpl.java:337 Comment()		

```
335         try {
336             //缁熶竴韬　唤涓　績鐧ュ綍涓撹瘉 - 瀵嗙爜楠岃瘉
337             //return gmcUserMapper.verifyAccountPassword(cUserId, password);
338             return 0;
339         } catch (Exception e) {
```

User.java, line 103 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	User.java:103 Comment()		

```
101
102 //    public void eraseCredentials() {
103 //        this.password =******
104 //    }
```

PassWordUtil.java, line 72 (Password Management: Password in Comment)

Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	PassWordUtil.java:72 Comment()		

```
70      }
```

```
71
72          /**
73           * 瑙ｅ瘡瀵嗙爜
74           * @param strPasswd
```

HttpRequest.java, line 2384 (Password Management: Password in Comment)			
Fortify Priority:	Low	**Folder**	
Kingdom:	Security Features		
Abstract:	以明文形式在系统或系统代码中存储密码或密码详细信息可能会以无法轻松修复的方式危及系统安全。		
Sink:	HttpRequest.java:2384 Comment()		

```
2382      }
2383
2384      /**
2385       * Set the 'Authorization' header to given values in Basic auth
2386       * format
```

Category: Unsafe Reflection (1 Issues)

Explanation:

若攻击者可以为应用程序提供确定实例化哪个类或调用哪个方法的参数值，那么就有可能创建一个贯穿于整个应用程序的控制流路径，而该路径并非是应用程序开发者最初设计的。这种攻击途径可能使攻击者避开 authentication 或 access control 检测，或使应用程序以一种意想不到的方式运行。即使狡猾的攻击者只能控制传送给指定函数或构造函数的参数，也有可能会成功地发起攻击。

如果攻击者能够将文件上传到应用程序的类路径或者添加应用程序类路径的新入口，那么对应用程序来说，情况会非常糟糕。无论处于上面哪种情况，攻击者都能通过反射将新的行为引入应用程序，而这一行为往往可能是恶意的。

示例：应用程序使用反射 API 的一个共同理由是实现自己的命令发送器。以下例子显示了一个没有使用反射的命令发送器：

```
String ctl = request.getParameter("ctl");
Worker ao = null;
if (ctl.equals("Add")) {
ao = new AddCommand();
} else if (ctl.equals("Modify")) {
ao = new ModifyCommand();
} else {
throw new UnknownActionError();
}
ao.doAction(request);
```

程序员可能会修改这段代码，以便按照如下情况使用反射：

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.doAction(request);
```

乍一看，这种修改似乎具有许多优点。代码的行数比原先少了，if/else 代码段也完全删除了，而且还可以在不改变命令发送器的情况下增加新的命令类型。

然而，这种修改允许攻击者将任意实现了 Worker 接口的对象实例化。如果命令发送器仍对 access control 负责，那么只要程序员创建实现 Worker 接口的新类，就务必要修改发送器的 access control 代码。如果未修改 access control 代码，那么一些 Worker 类就没有任何 access control 权限。

解决 access control 问题的方法是让 Worker 对象负责执行 access control 检查。下面是一段重新修改的代码：

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.checkAccessControl(request);
ao.doAction(request);
```

虽然有所改进，但它鼓励了采用分散化的手段进行 access control，这使程序员在 access control 上更加容易犯错误。

这段代码还强调了通过反射构建命令发送器所引发的安全问题。攻击者能为任意种类的对象调用默认构造函数。实际上，攻击者甚至不会局限于使用实现了 Worker 接口的对象；系统中所有对象的默认构造函数都可以调用。如果对象没有实现 Worker 接口，则会在分配 ao 前抛出 ClassCastException。但如果构造函数执行了一些有利于攻击者的操作，就会对应用程序造成伤害。对于简单的应用程序来说，这种情况的影响并不大，但是对于日趋复杂的大型应用程序来说，攻击者利用构造函数发动攻击并非没有可能。

如果在由反射调用返回的不可信任对象上调用使用即时调用者的类加载器检查执行任务的特定 Java API，则可能会危害访问检查，进而影响代码执行链。这些 Java API 会绕过可确保执行链中的所有调用者具有必需安全权限的 SecurityManager 检查。由于此类 API 可以绕过安全访问检查，使系统容易受到远程攻击，因此应格外小心，确保不要在由反射返回的不可信任对象上调用这些 API。有关这些 Java API 的更多信息，请参见“ Secure Coding Guidelines for the Java Programming Language ”中的准则 9。

Recommendations:

防止 unsafe reflection 的最佳方法是采用一些间接手段：创建一个规定用户使用的合法名称列表，并仅允许用户从中进行选择。通过这个方法，就不会直接采用用户提供的输入指定传输到反射 API 的名称。

反射也可以用来创建自定义的数据驱动体系结构，这样，配置文件就可以决定应用程序所使用的对象类型和组合。这种编程风格会带来以下安全问题：

— 控制程序的配置文件是程序源代码的重要组成部分，必须进行保护及相应的检查。

— 因为应用程序的配置文件是唯一的，所以必须执行独特的操作来评估设计的安全性。

— 因为当前应用程序的语意由一个自定义格式的配置文件支配，为了得出最理想的静态分析结果，就需要自定义一些规则。

鉴于以上原因，除非开发组可以在安全评估方面投入大量的精力，否则避免使用这种风格的设计。

Tips:

1. 在未使用用户输入调用反射方法前，要特别小心验证该用户输入的各种企图。因为应用程序很有可能比操作系统、file system 或其他系统模块发展得更快，因此，相对于将用户数据传送到其他系统模块所需的输入验证而言，用户输入验证所涉及的工作必须要走在它的前面。即使当前验证正确，也不能保证它在以后仍然正确。

2. 许多现代 Web 框架都提供对用户输入执行验证的机制。其中包括 Struts 和 Spring MVC。为了突出显示未经验证的输入源，HPE Security Fortify 安全编码规则包会降低 HPE Security Fortify Static Code Analyzer（HPE Security Fortify 静态代码分析器）报告的问题被利用的可能性，并在使用框架验证机制时提供相应的依据，以动态重新调整问题优先级。我们将这种功能称之为上下文敏感排序。为了进一步帮助 HPE Security Fortify 用户执行审计过程，HPE Security Fortify 软件安全研究团队提供了数据验证项目模板，该模板会根据应用于输入源的验证机制，将问题分组到多个文件夹中。

SecurityContextHolder.java, line 37 (Unsafe Reflection)

Fortify Priority:	Low **Folder**
Kingdom:	Input Validation and Representation
Abstract:	攻击者可以控制 SecurityContextHolder.java 中第 37 行的反射方法 forName() 所使用的参数，通过此种方式，创建一个意想不到且贯穿于整个应用程序的控制流路径，从而规避潜在的安全检查。

Source: SecurityContextHolder.java:12 java.lang.System.getProperty()

```
10    public static final String MODE_GLOBAL = "MODE_GLOBAL";
11    public static final String SYSTEM_PROPERTY = "spring.security.strategy";
12    private static String strategyName = System.getProperty("spring.security.strategy");
13    private static SecurityContextHolderStrategy strategy;
14    private static int initializeCount = 0;
```

Sink: SecurityContextHolder.java:37 java.lang.Class.forName()

```
35        } else {
36            try {
37                Class clazz = Class.forName(strategyName);
38                Constructor customStrategy = clazz.getConstructor(new Class[0]);
39                strategy = (SecurityContextHolderStrategy) customStrategy.newInstance(new Object[0]);
```