1/弱口令问题，代码虽然已经修改，但是数据库中的实际密码还是原来的，数据库中还存在弱口令的情况，易被破解。要将数据库中存在的弱口令用户，也修改过来。

信息中心建议：技术层面+人为管理层面双重限制。技术层面：限制达到一定时间未登陆过的用户，系统自动冻结，待企业登陆后联系相关人员进行解冻或重新申请账号；人为管理层面：通知各企业及时登陆修改密码。

2/验证码 针对暴力破解无效，前台只在输入用户名或密码错误情况下，刷新前台页面，才会生成新验证码，如果不通过前台，用工具破解，后台无法自动刷新验证码。

信息中心建议：通过后台收到一次请求即刷新一次验证码。

3/文件下载 只做文件后缀名限制，使用工具进行入侵的话，还是可以下载其他乃至上层路径的改类型后缀名的文件，应限制为只能下载自己上传的图片。

4/其他问题 当点击附件，删除时，会显示文件路径，在路径上随便改几个字符网站报错，在报错信息里会泄露很多东西，例如表名，服务器信息等..

如下图：



信息中心建议：网站报错后，返回统一报错页面，关键信息不要反馈到前台。

5/数据库 使用的用户权限太高，sa用户，换一个低权限用户。

6/SQL注入

针对SQL注入漏洞整改建议

1、采用预编译方式针对每个拼接参数修改成预编译的方式从根本上解决注入问题，针对使用ORM框架如Mybatis尽量使用“#”传参，针对一些order by 特殊要求建议针对此参数单独过滤。

2、 过滤方式针对每个提交信息的客户端页面，通过服务器端脚本（JSP、ASP、ASPX、PHP等脚本）生成的客户端页面，提交的表单（FORM）或发出的连接请求中包含的所有变量，必须对变量的值进行检查。过滤其中包含的特殊字符，或对字符进行转义处理。

严格定义应用程序可接受的数据类型（ 例如，字符串、字母数字字符等）。使用肯定的定义而非否定的定义。验证输入中是否存在不正确的字符。采用这样一种基本原理：使用肯定的定义而非否定的定义。不要向最终用户显示提供的信息（如表名）可用于策划攻击的错误消息。定义受允许的字符集。

特殊字符包括：

SQL语句关键词：如 and 、or 、select、declare、update、xp\_cmdshell；

SQL语句特殊符号：’、”、;等；

3、应用数据库权限要求：

严禁使用DBA权限作为应用所使用的链接账户，用户角色应遵循最小权限原则。

应用所使用的用户关闭执行系统命令、查询系统表功能。

当前用户权限完全控制在此应用所使用的数据表范围内。

7/跨站脚本漏洞

整改建议

每个提交信息的客户端页面，通过服务器端脚本（JSP、ASP、ASPX、PHP等脚本）生成的客户端页面，提交的表单（FORM）或发出的连接请求中包含的所有变量，必须对变量的值进行检查。过滤其中包含的特殊字符，或对字符进行转义处理。特殊字符包括：

HTML标签的<符号、“符号、’符号、%符号等，以及这些符号的Unicode值；

客户端脚本（Javascript、VBScript）关键字：javascript、script等；

此外，对于信息搜索功能，不应在搜索结果页面中回显搜索内容。同时应设置出错页面，防止Web服务器发生内部错误时，错误信息返回给客户端。具体建议如下：

1.   定义允许的行为。确保 Web 应用程序根据预期结果的严格定义来验证所有输入参数（Cookie、标头、查询字符串、表单、隐藏字段等）。

2.   检查POST 和 GET 请求的响应，以确保返回的对象是预期的内容且有效。

3.   通过对用户提供的数据进行编码，从用户输入中移除冲突的字符、括号和单双引号。这将防止将插入的脚本以可执行的格式发送给最终用户。

4.   只要可能，就应将客户端提供的所有数据限制为字母数字数据。使用此过滤机制时，如果用户输入“<script>alertdocumentcookie( 'aaa') </script>”，将缩减为“scriptalertdocumentcookiescript”。如果必须使用非字母数字字符，请先将其编码为 HTML 实体，然后再将其用在 HTTP 响应中，这样就无法将它们用于修改 HTML 文档的结构。

5.   使用双因素客户身份验证机制，而非单因素身份验证。

6.   在修改或使用脚本之前，验证脚本的来源。

7.   不要完全信任其他人提供的脚本并用在您自己的代码中（不论是从 Web 上下载的还是熟人提供给您的）。

8/所需出具相关报告

1自查报告

2再整改的情况报告