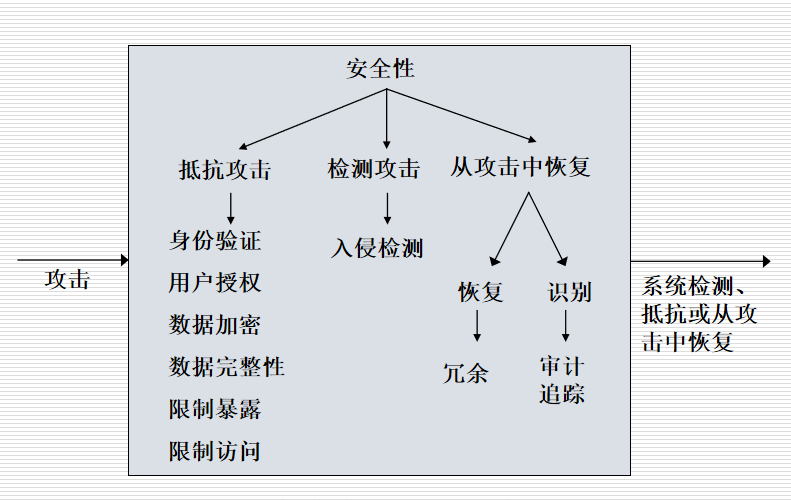
**软件体系架构质量属性之安全性性**

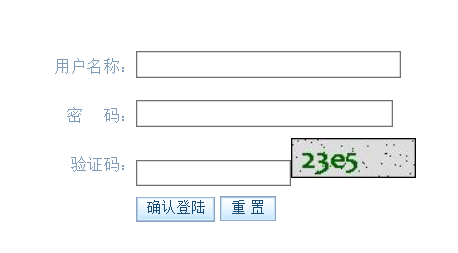
**时间：2020-03-02**

安全性的战术分为：与抵抗攻击有关的战术、与检测攻击有关的战术以及 与从攻击中恢复有关的战术。这3种战术都非常重要。我们对此做个很熟悉的比较，给 门装锁就是在抵抗攻出，在房子中放-个运动传感器就是在检测攻击.给房子上保险就是 从攻击中恢复过来。

**一．抵抗攻击**

1.身份验证(Authenticate users )：保证用户的合法身份，如口令、密码，生物识别。

  我在热词分析系统里用到了抵抗攻击的身份验证等，用这个系统，首先你的先登录账号



 并且还得输入验证码，通过session传值到servlet来验证

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<%@ page import="java.util.Random"%>

<%@ page import="java.io.OutputStream"%>

<%@ page import="java.awt.Color"%>

<%@ page import="java.awt.Font"%>

<%@ page import="java.awt.Graphics"%>

<%@ page import="java.awt.image.BufferedImage"%>

<%@ page import="javax.imageio.ImageIO"%>

<%

int width = 100;

int height = 32;

//create the image

BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);

Graphics g = image.getGraphics();

// set the background color

g.setColor(new Color(0xDCDCDC));

g.fillRect(0, 0, width, height);

// draw the border g.setColor(Color.black);

g.drawRect(0, 0, width - 1, height - 1);

// create a random instance to generate the codes

Random rdm = new Random();

String hash1 = Integer.toHexString(rdm.nextInt());

// make some confusion

for (int i = 0; i < 50; i++) {

int x = rdm.nextInt(width);

int y = rdm.nextInt(height);

g.drawOval(x, y, 0, 0);

}

// generate a random code

String capstr = hash1.substring(0, 4);

session.setAttribute("key11", capstr);//将验证码存储到session中

g.setColor(new Color(0, 100, 0));

g.setFont(new Font("Candara", Font.BOLD, 24));

g.drawString(capstr, 8, 24);

g.dispose();

response.setContentType("image/jpeg");

out.clear();

out = pageContext.pushBody();

OutputStream strm = response.getOutputStream();

ImageIO.write(image, "jpeg", strm);

strm.close();%>

然后再servlet 接受String code1=(String) session.getAttribute("key11"); 来验证

2.维护数据的机密性(Maintain data confidentiality )：对传输数据进行加密

然后对于注册登录的密码进行了md5加密。

package servlet;

import java.security.MessageDigest;

/\*\*

\* Created by geely

\*/public class md5 {

private static String byteArrayToHexString(byte b[]) {

StringBuffer resultSb = new StringBuffer();

for (int i = 0; i < b.length; i++)

resultSb.append(byteToHexString(b[i]));

return resultSb.toString();

}

private static String byteToHexString(byte b) {

int n = b;

if (n < 0)

n += 256;

int d1 = n / 16;

int d2 = n % 16;

return hexDigits[d1] + hexDigits[d2];

}

/\*\*

\* 返回大写MD5

\*

\* @param origin

\* @param charsetname

\* @return

\*/

private static String MD5Encode(String origin, String charsetname) {

String resultString = null;

try {

resultString = new String(origin);

MessageDigest md = MessageDigest.getInstance("MD5");

if (charsetname == null || "".equals(charsetname))

resultString = byteArrayToHexString(md.digest(resultString.getBytes()));

else

resultString = byteArrayToHexString(md.digest(resultString.getBytes(charsetname)));

} catch (Exception exception) {

}

return resultString.toUpperCase();

}

public static String MD5EncodeUtf8(String origin) {// origin = origin + PropertiesUtil.getProperty("password.salt", "");

return MD5Encode(origin, "utf-8");

}

private static final String hexDigits[] = {"0", "1", "2", "3", "4", "5",

"6", "7", "8", "9", "a", "b", "c", "d", "e", "f"};

}

通过调用这个方法来对输入的密码进行加密 ，然后传到dao层，进行注册登录验证。而且dao层的连接数据库用到了这个PreparedStatement可以防止登录账号的时候输入一些符号，来消除sql语句里的where 后面的条件这样子可以保障了账号密码的安全性，防止他人窃取密码。

3.限制访问，防火墙根据消息源或目的地端口来限制访问。来自未知源的消息可能 是某种形式的攻击。限制对已知源的访问并不总是可行的。例如，一个公共网站 上可能会有来自未知源的请求。这种情况中使用的一个配置就足所谓的解除管制 区（DMZ)。当必须对Internet服务而非专有网提供访问时使用DMZ。它位于 Internet和内部网前面的防火墙之间。DMZ包含预计会从任意源接收消息的设备， 这些信息源包括Web服务、e-mail和域名服务等。我在jsp的界面上用到了session 将登录成功的账号存入了session里，在其他jsp上进行session验证，查看是否登录了账号，如果没有登录账号，将无法在所有界面查看。

<%

if((String)session.getAttribute("username")==null){

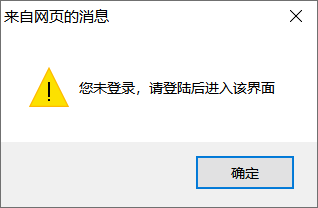
%>

<script type="text/javascript">

alert("您未登录，请登陆后进入该界面");

</script>

<%} %>



 可以防止知道主功能功能网页后未经过登录来直接进入网页上来。

    用户授权在这里没有用到，但之前有过账号分为管理员账户和普通账号，这两个账号登陆上去的权限不一样

**二．检测攻击**

检测攻击的工作方式通常是将网络通信模式与数据库中已知攻击的历史记录进行对比； 入侵检验一般需要对数据包进行过滤，以进行比较； 入侵检验必须有检测攻击的传感器，并且可以存贮事件到数据库中，供以后数据分析使用

**三．从攻击中恢复**

从攻击中恢复的战术包括与恢复状态相关的战术和识别攻击者相关的战术。

1.与恢复状态相关的战术与可用性一致，如冗余，但主要是对密码、访问控制列表和用户资料数据进行冗余。

2.

用于识别攻击者的战术是“维持审计追踪”审计追踪就是应用到系统中的数据的所有事物和识别信息的一个副本，可以使用它来识别攻击者的操作，支持认可和系统恢复。