简单写一下审计了几个月这个 oa 的思路
首先肯定还是看 web.xml
对 jsp 文件的鉴权以及 oa 的补丁都在 SecurityFilter 中

```
<filter>
    <filter-name>SecurityFilter</filter-name>
    <filter-class>weaver.filter.SecurityFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>SecurityFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<filter>
```

E8 和 E9 的主要差别就是 E9 多了一个/api 的访问 也就是这个 servlet

```
<servlet>
    <servlet-name>restservlet</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>com.cloudstore;com.api</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>restservlet</servlet-name>
    <url-pattern>/api/*</url-pattern>
</servlet-mapping>
```

这个 servlet 就是扫描 com.cloudstore 和 com.api 包下的@Path 注解的一个 rest 接口
SecurityFilter 初始化时会调用 weaver.security.filter.SecurityMain 的 initFilterBean 方法会初始化安全规则 会加载这两个文件夹下对应 Ecology 版本的 XML 规则文件

```
new Thread(run() -> {
    List<String> files = new ArrayList();
    SecurityCore.this.listFiles(files, dirName: SecurityCore.access$0() + "WEB-INF" + File.separatorChar + "securityRule" + File.separatorChar + "Rule");
    List<String> filesx = SecurityCore.access$1(SecurityCore.this, files);
    SecurityCore.this.writeLog( log: "files:::::" + filesx, isErrMsg: true);
    if ("true".equals(SecurityCore.this.null2String(SecurityCore.this.getRule().get("init-field-from-db"))) || SecurityCore.this.null2String(SecurityCore
        try {
            try {
                Thread.currentThread();
                Thread.sleep( millis: 5000L);
            } catch (InterruptedException var3) {
            }

            SecurityCore.this.initCustomFieldFromDB();
        } catch (Throwable var4) {
            SecurityCore.this.writeError(var4);
        }
    }

    SecurityCore.this.init(filesx, (String)null);
    filesx.clear();
    SecurityCore.this.listFiles(filesx, dirName: SecurityCore.access$0() + "WEB-INF" + File.separatorChar + "securityXML");
    SecurityCore.this.init(filesx, (String)null);
    SecurityCore.this.getRule().put("securityRuleInitComplete", "true");
```

后面这些代码太多了 就不一一去看 一个重要的点就是会去调用 weaver.security.rules.ruleImp 包下面的每一个类的 validate 方法 这些类差不多就是 E9 每次打的补丁 我们可以在拿到补丁后直接看到这些洞的 URL 这里有一个小 tips 就是如果是因为安全规则访问某个点导致的 404 在返回的 header 里会有一个 securityIntercept

```
Content-Length: 534
Content-Type: text/html; charset=utf-8
Date: Tue, 19 Jul 2022 13:02:58 GMT
errorMsg: securityIntercept
Server: WVS
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1
```

请求标头

泛微在这几个月修复了一系列由于 imagefilemanager 导致的 RCE 漏洞
我们先看到这次某微在 3 月份打的一个补丁
weaver.security.rules.ruleImp.SercurityRuleForCptInventory
这个补丁针对性的修复了我交给某平台的一个前台 RCE 漏洞 并且只是在我交洞的一个月后 我没拿洞去打 也没给过任何人 所以 懂得都懂 ~

```
ist<String> allowTypes = new ArrayList();
llowTypes.add(".xls");
llowTypes.add(".xlsx");
ecurityCore sc = new SecurityCore();

ry {
   String path = sc.null2String(req.getRequestURI()).toLowerCase().trim();
   path = super.path(path);
   User user = sc.getUser(req);
   if ("E9".equals(sc.getEcVersion())) {
       boolean result = true;
       ImageFileManager manager;
       int fileid;
       String filename;
       if (path.contains("/api/") && path.contains("/cpt/") && path.contains("/inventory/") && path.contains("/docptimpoptinventory")) {
           manager = new ImageFileManager();
           fileid = Util.getIntValue(Util.null2String(req.getParameter( s: "inventoryexcelfile")),  i: 0);
           manager.getImageFileInfoById(fileid);
           filename = manager.getImageFileName();
           result = FileType.validateFileExt(filename, allowTypes);
```

路径是/api/cpt/inventory/docptimpoptinventory
这里拿到 inventoryexcelfile 参数调用 getImageFileInfoById 方法拿到 ImageFileManager 随后调用 getImageFileName 验证了后缀名
我习惯在审计的时候把这些库文件丢到 jd-gui 里面全部反编译出来方便在 idea 里面搜索
在 com.engine.cpt.web.CptInventoryAction 类里 跟进

```
@POST
@Path("/doCptimpoptInventory")
@Produces({"text/plain"})
public String doCptimpoptInventory(@Context HttpServletRequest var1, @Context HttpServletResponse var2) {
    User var3 = HrmUserVarify.getUser(var1, var2);
    Map var4 = ParamUtil.request2Map(var1);
    Map var5 = this.getService(var3).doCptimpoptInventory(var3, var4, var1);
    return JSONObject.toJSONString(var5);
}
```

```
try {
    ImageFileManager var8 = new ImageFileManager();
    int var34 = Util.getIntValue(Util.null2String(this.params.get("inventoryexcelfile")), 0);
    var8.getImageFileInfoById(var34);
    String var9 = var8.getImageFileName();
    String var10 = var8.getFileRealPath();
    String var11 = var8.getIsencrypt();
    String var12 = var8.getAescode();
    if (!var10.equals("")) {
        var6 = GCONST.getRootPath() + "cpt" + File.separatorChar + "ExcelToDB" + File.separatorChar + var9;
        InputStream var15 = null;
        FileOutputStream var16 = null;

        try {
            var15 = var8.getInputStream();
            if (var11.equals("1")) {
                var15 = AESCoder.decrypt(var15, var12);
            }

            var16 = new FileOutputStream(var6);
            byte[] var13 = new byte[1024];

            int var14;
            while((var14 = var15.read(var13)) != -1) {
                var16.write(var13, off: 0, var14);
            }
        } catch (Exception var31) {
            var31.printStackTrace();
        } finally {
```

可以发现这里是拿到 inventoryexcelfile 参数然后查询数据库后将 imagefilename 拼接到了路径中直接将文件写入到/cpt/ExcelToDB/目录中　所以我们现在需要找到一个前台的地方进行上传 jsp 木马写入 imagefile 表中并且还要回显 imagefileid 当我把 e9 源码熟悉之后 直接在所有 jsp 文件中搜索 new FileUpload(request,"utf-8") 由此找到了这个文件可以前台访问 workrelate/plan/util/uploaderOperate.jsp 这个文件可以达到上面的要求

```
int userid=Util.getIntValue(desUtil.decrypt(fu.getParameter("userid")),0);
int language=Util.getIntValue(fu.getParameter("language"),0);
int logintype=Util.getIntValue(fu.getParameter("logintype"),0);
int departmentid=Util.getIntValue(fu.getParameter("departmentid"),0);
User user=new User();
user.setUid(userid);
user.setLanguage(language);
user.setLogintype(""+logintype);
user.setUserDepartment(departmentid);

int mainId=Util.getIntValue(fu.getParameter("mainId"),0);
int subId=Util.getIntValue(fu.getParameter("subId"),0);
int secId=Util.getIntValue(fu.getParameter("secId"),0);
String plandetailid=Util.getIntValue(fu.getParameter("plandetailid"),0)+"";

StringBuffer restr = new StringBuffer();
if(secId!=0){
    int docid=deu.uploadDocToImg(fu,user, "Filedata",mainId,subId,secId,"","");
    if(!"0".equals(plandetailid)){
```

跟进这个方法

```
public int uploadDocToImg(FileUpload var1, User var2, String var3, int var4, int var5, int var6, String
    String var9 = "";

    try {
        SecCategoryComInfo var10 = new SecCategoryComInfo();
        DocComInfo var11 = new DocComInfo();
        DocManager var12 = new DocManager();
        DocViewer var13 = new DocViewer();
        DocImageManager var14 = new DocImageManager();
        int var15 = Util.getIntValue(var1.getParameter( s: "workflowid"),  i: 0);
        String var16 = "0";
        if (var15 > 0) {
            RequestDoc var17 = new RequestDoc();
            ArrayList var18 = var17.getDocFiled( s: "" + var15);
            if (var18 != null && var18.size() > 6) {
                var16 = "" + var18.get(6);
            }
        }

        String var32 = "1";
        if ("1".equals(var16)) {
            var32 = "9";
        }

        int var33 = Util.getIntValue(var1.getParameter( s: "isFirstUploadFile"),  i: 0);
        if (var33 > 0 && !"9".equals(var32)) {
            var32 = "9";
        }

        RecordSet var19 = new RecordSet();
        int var20 = var12.getNextDocId(var19);
        String var21 = var1.uploadFiles(var3);
        if (Util.getIntValue(var21) == -1) {
```

关键点在这个 uploadFiles 方法

```
    try {
        var7 = URLDecoder.decode(var7,  enc: "utf-8");
    } catch (Exception var36) {
        var7 = var7;
    }
}

String var23 = "" + var38 + var20 + var7 + var20 + var8 + var20 + var39 + var20 + var14 + var20 + var17 + var20 + var18 + var20 + var9;
var19.executeProc( s: "ImageFile_Insert", var23);
AliOSSObjectManager var24 = new AliOSSObjectManager();
```

跟进后就会发现这个类 weaver.file.FileUpload 的所有文件上传操作都会进行压缩之后写入到 imagefile 数据表中 然后将文件放到 filesystem 目录 并且在文件上传的过程中不会验证文件的后缀名 作为 imagefilename 字段写入到数据库中

到这里似乎已经可以写入 webshell 了 但是如果此时我们一步步的上传然后去访问会发现直接 404 了 这是因为有对应的规则 访问/cpt/ExcelToDB/目录中的 jsp 文件必须登陆才能访问 我们需要的是前台 RCE 所以必须得找一个前台的地方我们可以完全控制写入数据库中的 imagefilename 参数 使其为/../../xxx.jsp 在 weaver.file.FileUpload 类的上传中 调用的是某微自己写的文件上传解析类 weaver.file.multipart.MultipartParser

```
    }

    String name = origline.substring(start + startOffset, end);
    String filename = null;
    String origname = null;
    start = line.indexOf( str: "filename=\"",  fromIndex: end + 2);
    end = line.indexOf( str: "\"",  fromIndex: start + 10);
    if (start != -1 && end != -1) {
        filename = origline.substring(start + 10, end);
        origname = filename;
        int slash = Math.max(filename.lastIndexOf( ch: 47), filename.lastIndexOf( ch: 92));
        if (slash > -1) {
            filename = filename.substring(slash + 1);
        }
    }

    retval[0] = disposition;
    retval[1] = name;
    retval[2] = filename;
    retval[3] = origname;
    return retval;
}
else {
```

直接取最后一个斜杠后的内容为文件名 通过搜索 new ServletFileUpload 寻找调用 commons-fileupload 库来解析文件上传的地方 最后找到了 /api/fna/documentCompare/doCreateDocument 这个路径

```
        ServletFileUpload var32 = new ServletFileUpload(var30);
        List var33 = var32.parseRequest(var1);
        if (var33 == null) {
            var3.put("status", "-1");
            var3.put("errorInfo", "108 表单无数据");
            return var3;
        }

        JSONArray var8 = new JSONArray();
        JSONArray var9 = new JSONArray();
        var3.put("imagefileId", var9);
        var3.put("imagefileName", var8);
        int var10 = var33.size();
        var10 = Math.min(var10, 2);

        for(int var11 = 0; var11 < var10; ++var11) {
            FileItem var12 = (FileItem)var33.get(var11);
            if (!var12.isFormField()) {
                String var13 = var12.getName();
                var13 = var13.substring(var13.lastIndexOf( str: "\\") + 1);
                String var14 = "";
                int var15 = var13.lastIndexOf( str: ".");
                if (var15 != -1) {
                    var14 = var13.substring(var15 + 1).toLowerCase();
                }

                Object var16 = null;
                InputStream var17 = var12.getInputStream();
                ByteArrayOutputStream var18 = new ByteArrayOutputStream();
                byte[] var19 = new byte[1024];

                int var20;
```

这里可以将文件名原样的写入到 imagefile 表中 返回 imagefileid 具体的就不去跟了
最后还有一个点就是/api 路径的鉴权 我们上面所提到的这两个路径都是需要登陆后访问的 解决办法就是大写 /API/像这样访问就不需要登陆

```
    </filter>
    <filter-mapping>
        <filter-name>SessionCloudFilter</filter-name>
        <url-pattern>/api/*</url-pattern>
    </filter-mapping>
```

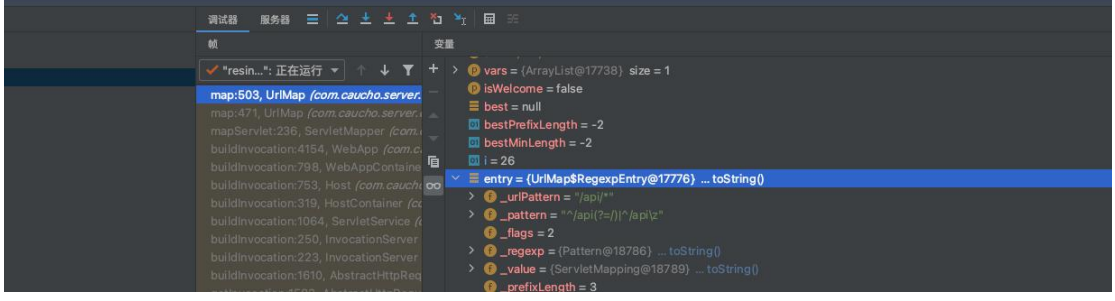
/api/路由的鉴权在这个类中 335 行

```
331         }
332
333         if (StringUtil.isNotNullAndEmpty(uncheckSessionUrl)) {
334             var35 = uncheckSessionUrl.split( regex: ";");
335             if (this.checkUrl(var4.getRequestURI(), "/api/") && !this.checkUrl(var4.getRequestURI(), var35)) {
336                 JSONObject var37 = this.getwrongJSONObject( var1: 2);
337                 var2.setContentType("application/json;charset=utf-8");
338                 var2.setCharacterEncoding("utf-8");
339                 var2.getWriter().println(var37.toString());
340                 logger.error( o: "超时拦截:" + var4.getRequestURI() + ",当前sessionId: " + var6.getId() + ",hasCookie:" + (O
341                 return;
342             }
343
344             var3.doFilter(var4, var5);
```

```
private boolean checkUrl(String var1, String var2) {
    return StringUtil.isNotNullAndEmpty(var1) && StringUtil.isNotNullAndEmpty(var2) && var1.startsWith(var2);
}
```

可以看到这里拿到 requesturi 之后并没有转换大小写 导致/API/可以绕过这个鉴权
那为什么大写 api 还能同样到达这个路由呢 tomcat 是不可以的 之前我以为这是 windows 和 linux 系统的差异导致的 但是后面发现是 Resin 导致的
调 resin 的过程可以看这里 https://blog.csdn.net/HBohan/article/details/121163220

```
for(int i = 0; i < this._regexps.size(); ++i) {  i: 26
    UrlMap.RegexpEntry<E> entry = (UrlMap.RegexpEntry)this._regexps.get(i);  entry: UrlMap$RegexpEntry@17776  i: 26
    if ((!isWelcome || entry.isSimple()) && !entry.isIgnore() && entry._prefixLength >= bestPrefixLength) {  bestPrefixL
        Matcher matcher = entry._regexp.matcher(uri);  uri: "/API/axx"  entry: UrlMap$RegexpEntry@17776  _regexp: Pat
        if (matcher.find()) {
            int begin = matcher.start();
            int end = matcher.end();
            int length = end - begin;
            if (bestPrefixLength < entry._prefixLength || bestMinLength < length) {
                if (vars != null) {
                    vars.clear();
                    if ("/*".equals(entry.getPattern())) {
                        vars.add("");
                    } else {
                        vars.add(uri.substring(0, end));
                    }
```



这里的 urlpattern 正则为^/api(?=/)|^/api\z 大小写不敏感 所以仍然可以匹配到这个 servlet
后续某微修复了大写 api

```
            return false;
        }
    } else {
        if (path.contains("//")) {
            path = path.replaceAll( regex: "/{2,}", replacement: "/");
        }

        List<String> paths = new ArrayList();
        paths.add("/API/");
        paths.add("/APi/");
        paths.add("/Api/");
        paths.add("/aPI/");
        paths.add("/aPi/");
        paths.add("/apI/");
        paths.add("/ApI/");
        boolean result = true;
        Iterator var8 = paths.iterator();

        while(var8.hasNext()) {
            String p = (String)var8.next();
            if (path.startsWith(p)) {
                result = false;
            }
        }
```

至此后面少了很多的前台 rce.....
本文只是抛砖引玉，不对的地方希望师傅们指点指点。