

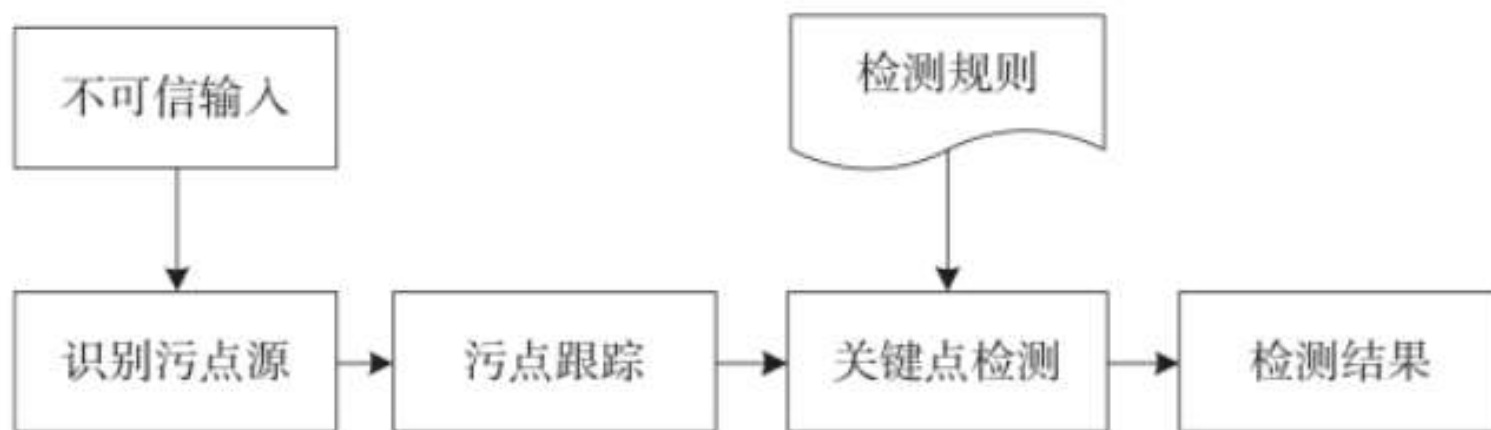
Firmeye固件漏洞挖掘插件

Firmeye固件漏洞挖掘插件



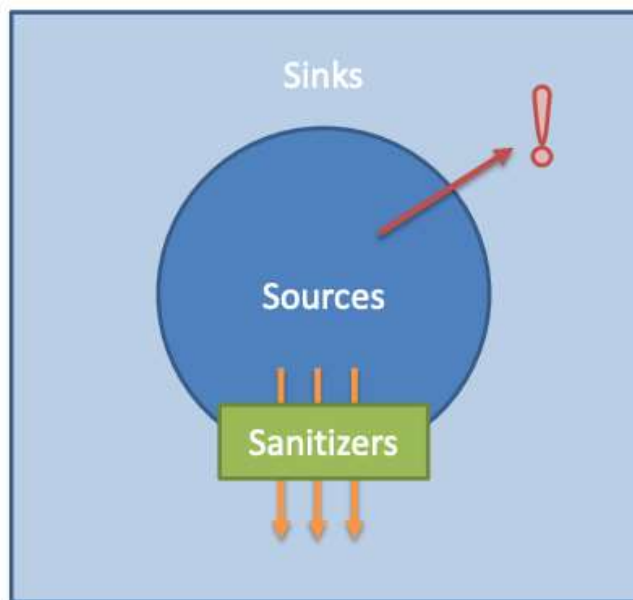
污点分析

- 污点分析就是分析程序中由污点源（source）引入的数据是否能够不经无害处理，而直接传播到污点汇聚点（sink）。如果不能，说明系统是数据流安全的；否则，说明系统产生了隐私数据泄露或危险数据操作等安全问题。

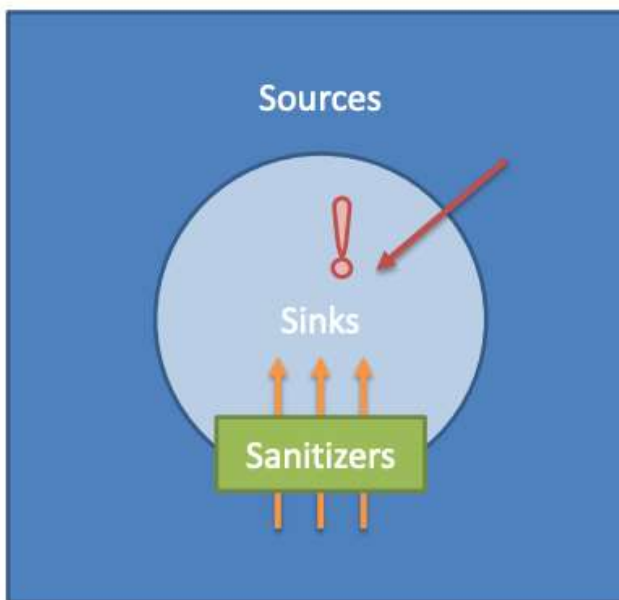


污点分析

- 污点分析关键点可以抽象成一个三元组：
 $\langle \text{sources}, \text{sinks}, \text{sanitizers} \rangle$
- sources**: 即污点源，代表直接引入的不受信任的数据
- sinks**: 即汇聚点，代表污点检测点，判断是否有被污染的数据到达汇聚点
- sanitizers**: 即无害化处理，代表通过数据加密或者移除危害操作等手段使数据传播不再对软件系统的信息安全产生危害



隐私数据泄漏



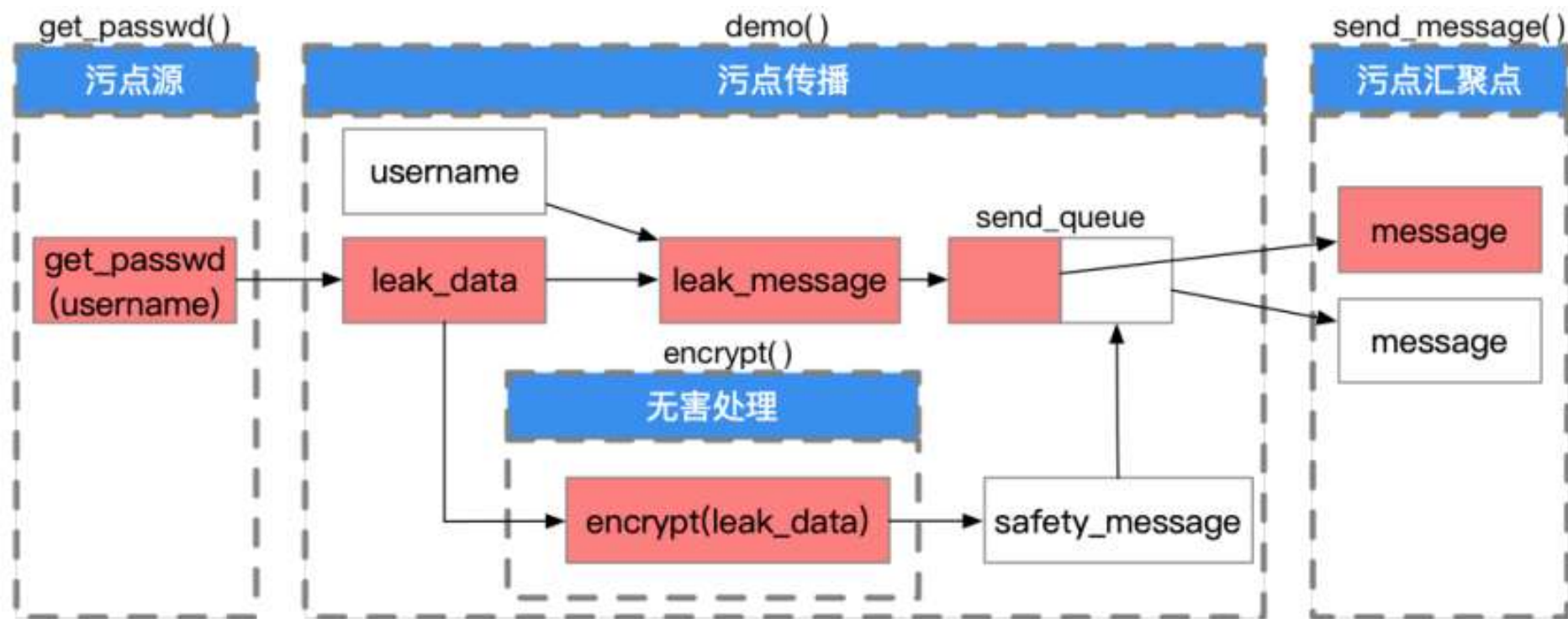
检测危险数据操作

污点分析

- 污点分析的处理过程
 - (1) 识别污点源：判定哪些输入或数据是污点
 - (2) 划定汇聚点：划定在哪些区域检测污点
 - (3) 污点传播分析：将受影响的其他变量\数据\寄存器\内存等添加污点标记
 - (4) 无害化处理：某些变量\数据\寄存器\内存等经过处理后不再是污点，去掉其污点标记
 - (5) 汇聚点检测：检测某些函数输入输出\变量\寄存器\内存等处是否存在污点标记
 - (6) 异常处理：汇聚点检测出污点后，执行指定代码
- 步骤 (1) (2) 是前提，步骤 (3) ~ (5) 无顺序关系

污点分析

```
1 def demo(username):
2     send_queue = []
3     leak_data = get_passwd(username)           # source
4     leak_message = 'Username=' + username + '|passwd=' + str(leak_data)
5     send_queue.append(leak_message)
6     safety_message = encrypt(leak_message)      # sanitizers
7     send_queue.append(safety_message)
8     for message in send_queue:
9         send_message(message)                  # sinks
```



污点分析

- 显示流污点传播：污点数据直接参与数据流传播，例如：直接赋值传播、通过函数（过程）调用传播以及通过别名（指针）传播
- 隐式流污点传播：污点数据间接控制数据流传播，主要通过控制依赖进行传播：污点数据作为程序分支的判断条件

```
11  def demo2(pwd):                # source
12      data = ''
13      if pwd == '123456':
14          data = get_secret()
15      else:
16          data = get_fake()
17      return data                 # sink
18
```

污点分析

- 静态分析技术：

- 特点：1. 不运行代码 2. 不修改代码
- 方法：通过分析程序变量间的依赖关系来实现
- 优点：速度快，能够快速定位污点在程序中可能出现的传播路径
- 缺点：精度低，因为无法获取程序运行时的一些必要信息，分析结果往往存在偏差，需要人工对分析结果进行复查

- 动态分析技术：

- 特点：1. 在程序运行时进行分析 2. 只分析运行时可达的支路
- 方法：通过实时监控程序的污点数据在系统程序中的传播来实现
- 优点：分析准确率较静态分析高
- 缺点：由于引入了代码重写和插桩等机制，会显著增加程序的开销

敏感函数参数回溯

- 将污点分析的过程反过来，从Sink点（敏感函数）出发寻找Source点（函数参数入口）

```
1 __int64 sub_4007B0()  
2 {  
3     char format; // [rsp+0h] [rbp-20h]  
4     unsigned __int64 v2; // [rsp+18h] [rbp-8h]  
5  
6     v2 = __readfsqword(0x28u);  
7     puts("Good !!! Level up~ :");  
8     __isoc99_scanf("%10s", &format);  
9     printf(&format);  
10    return 0LL;  
11 }
```

int scanf(const char *format, ...);

int printf(const char *format, ...);

```
push    rbp  
mov     rbp, rsp  
sub     rsp, 20h  
mov     rax, fs:28h  
mov     [rbp+var_8], rax  
xor     eax, eax  
mov     edi, offset s ; "Good !!! Level up~ :"  
call    _puts  
lea     rax, [rbp+format]  
mov     rsi, rax  
mov     edi, offset a10s ; "%10s"  
mov     eax, 0  
call    __isoc99_scanf  
lea     rax, [rbp+format]  
mov     rdi, rax ; format  
mov     eax, 0  
call    _printf  
mov     eax, 0  
mov     rdx, [rbp+var_8]  
xor     rdx, fs:28h  
jz      short locret_400811  
call    __stack_chk_fail
```

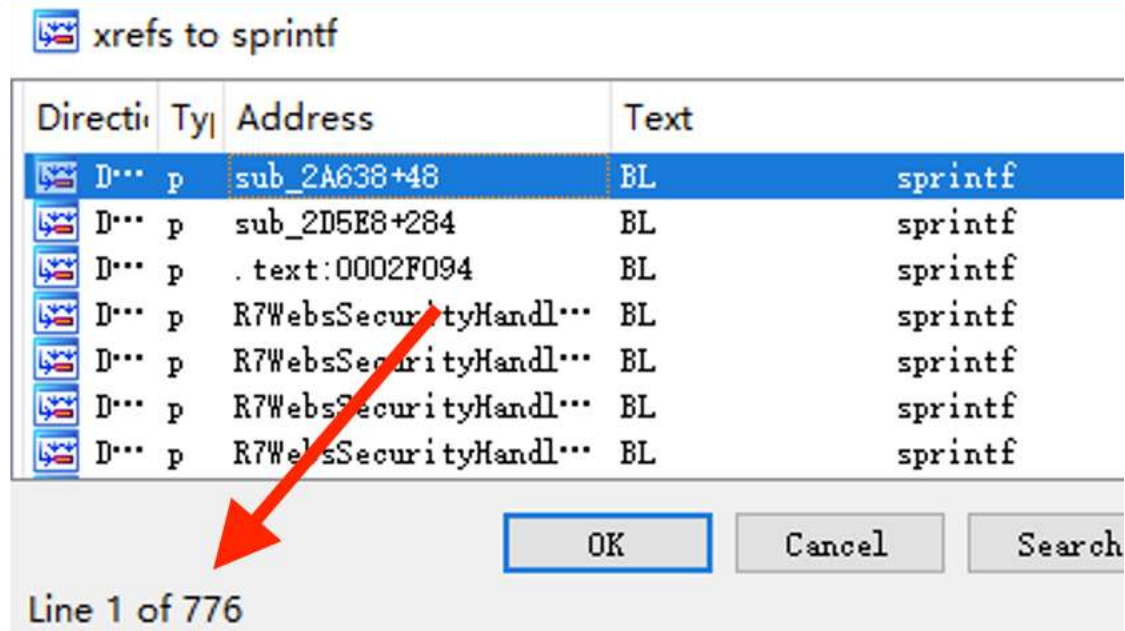
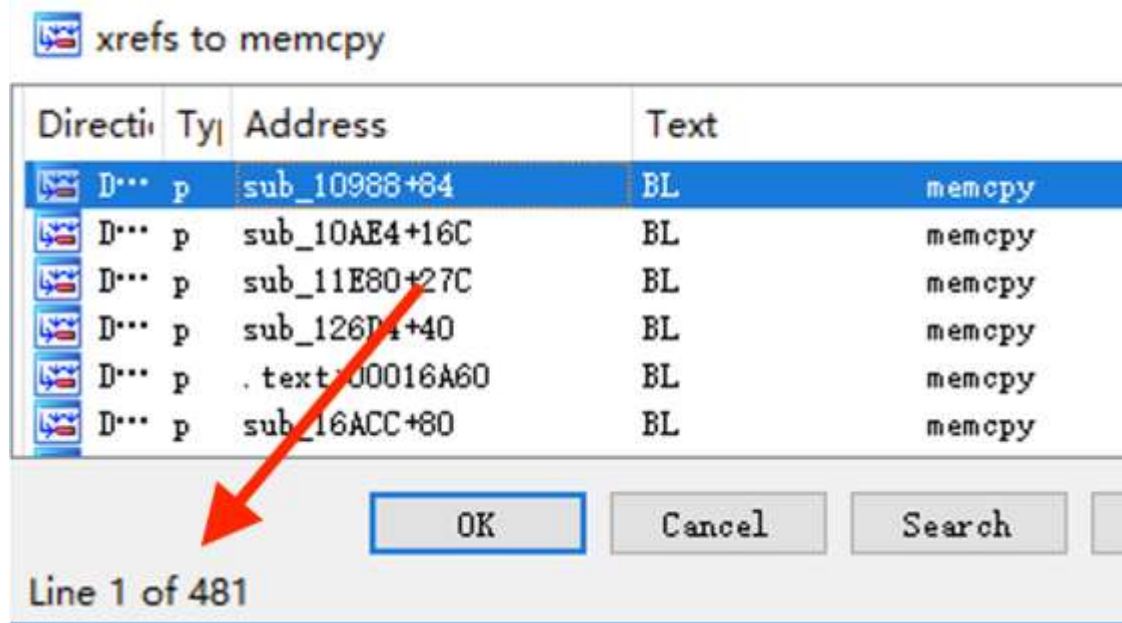

敏感函数参数回溯

- 二进制代码级的污点分析有两种粒度，一是直接指令，二是中间语言（IR）。直接指令是x86等指令集，中间语言指经过二进制代码翻译后的语言。

指令类型	传播规则	举例说明
拷贝或移动指令	$T(a) \leftarrow T(b)$	mov a, b
算数运算指令	$T(a) \leftarrow T(b)$	add a, b
堆栈操作指令	$T(esp) \leftarrow T(a)$	push a
拷贝或移动类函数调用指令	$T(dst) \leftarrow T(src)$	call memcpy
清零指令	$T(a) \leftarrow false$	xor a, a


注：T(x) 的取值分为 true 和 false 两种，取值为 true 时表示 x 为污点，否则 x 不是污点。

自动化IDA插件



自动化IDA插件 - 界面

```
##### FIRMEYE TOOLKITS #####
#
#       an auxiliary tool for iot vulnerability hunter
#
# ----- HOT KEAY -----
#
#   Ctrl+F1           show this help
#
# ----- STATIC ANALYZER -----
#
#   Ctrl+Shift+s      main menu
#
# ----- DYNAMIC ANALYZER -----
#
#   Ctrl+Shift+d      enable/disable debug hook
#
# ----- REVERSE ASSISTANT -----
#
#   Ctrl+Shift+x      reverse assist tools
#
# ----- FUNCTIONAL TEST -----
#
#   Ctrl+Shift+q      functional test
#
#####
```

 Firmeye Static Analyzer ✕

危险函数地址:

查看

对指定函数调用地址下断点:

添加断点

删除断点

仅添加函数

危险函数调用地址 (全部):

查看

添加断点

删除断点

危险函数调用地址 (指定):

查看

添加断点

删除断点

危险函数漏洞分析 (全部):

查看

添加断点

删除断点

危险函数漏洞分析 (指定):

查看

添加断点

删除断点

给所有断点的下一条指令下断点:

添加断点

添加并删除当前断点

导出/导入离线断点:

导出

导入

OK

Cancel

自动化IDA插件 - 回溯逻辑

```
def get_next_reg(self, addr, reg):
    reg_t = reg
    addr_t = addr

    mnemonic_t = idc.print_insn_mnem(addr_t)
    line = idc.generate_disasm_line(addr_t, 0)
    if reg_t == 'R0' and mnemonic_t.startswith('BLX') and addr_t != self.trace_addr:
        FirmEyeLogger.info("找到赋值点\t"+num_to_hexstr(addr)+"\t"+line)
        return None

    inst_list_t = INST_LIST
    reg_re = re.compile(reg_t + '\\D|' + reg_t + '\\Z')
    if reg_re.search(line):
        if mnemonic_t in reduce(lambda x, y: x + y, [value for value in inst_list_t.values()]):
            op1 = idc.print_operand(addr_t, 0).split("!")[0]
            if mnemonic_t in inst_list_t['load_multi']: ...
            else:
                if op1 != reg_t or mnemonic_t in inst_list_t['other']: ...
                elif mnemonic_t in inst_list_t['arithmetic']: ...
                elif mnemonic_t in inst_list_t['move']: ...
                elif mnemonic_t in inst_list_t['load']: ...
                else:
                    FirmEyeLogger.info("未知指令\t"+num_to_hexstr(addr)+"\t"+line)
        else:
            FirmEyeLogger.info("未知指令\t"+num_to_hexstr(addr)+"\t"+line)
```

自动化IDA插件 - 漏洞判断逻辑

```
for xref_addr_t in xref_list_t:
    FirmEyeLogger.info("从%s回溯来源地址%s" % (num_to_hexstr(xref_addr_t), vuln_reg))
    tracer = FirmEyeArgsTracer(xref_addr_t, vuln_reg)
    source_addr = tracer.run()
    print 'source_addr: ', source_addr
    # 判断是否找到目标地址
    if source_addr == []:
        FirmEyeLogger.info("目标地址未找到%s" % num_to_hexstr(xref_addr_t))
        vuln_flag = 1
    else:
        for cmd_addr in source_addr:
            addr1 = cmd_addr
            # 判断字符串是否来自内存
            if idc.get_operand_type(cmd_addr, 1) == idaapi.o_mem:
                cmd_str = FirmEyeStrMgr.get_mem_string(cmd_addr)
                # 判断是否找到字符串
                if cmd_str == []:
                    FirmEyeLogger.info("硬编码命令未找到%s" % num_to_hexstr(xref_addr_t))
                    vuln_flag = 1
                else:
                    vuln_flag = 0
                    str1 = cmd_str[0]
            else:
                FirmEyeLogger.info("命令来自外部%s" % num_to_hexstr(xref_addr_t))
                vuln_flag = 1
```

自动化IDA插件 - 漏洞分析

通过自动化分析插件，排除 80% 的无危险调用，大大提高了效率

危险函数地址

	函数名	函数地址
0	system	0x0000ed24
0	memcpy	0x0000ed6c
0	strcat	0x0000ee2c
0	snprintf	0x0000f000
0	strncat	0x0000f1ec
0	sscanf	0x0000f324
0	strncpy	0x0000f330
0	fprintf	0x0000f4a4
0	vsnprintf	0x0000f51c
0	sprintf	0x0000f54c
0	printf	0x0000f798
0	popen	0x0000f7a4
0	strcpy	0x0000f87c

Line 10 of 13

危险函数漏洞分析(快)

可疑	函数名	函数地址	格式	格式字符串
0	sprintf	0x000bb06c	0...	wan%d_check
0	sprintf	0x000bb0b0	0...	wan%d_action_flag
0	sprintf	0x000c0de8	0...	%d,
0	sprintf	0x000c27e0	0...	{"errCode":%d}
0	sprintf	0x000c3b40	0...	{"errCode":%d}
1	sprintf	0x0002a680	0...	/webroot%s
1	sprintf	0x00031838	0...	%s_%s
1	sprintf	0x0003bba4	0...	/system_backup.html?%s
1	sprintf	0x0003cba0	0...	cat /proc/kmsg >> %s &
1	sprintf	0x0003cc60	0...	wl -i %s %s
1	sprintf	0x0003cd40	0...	wl -i %s %s
1	sprintf	0x0003d908	0...	%s=%s
1	sprintf	0x0003db64	0...	%s=%s

Line 542 of 665

CVE-2020-13392

- Tenda路由器的Web服务器httpd中存在一个缓冲区溢出漏洞，在处理post请求的funcpara1的参数时，sprintf将该值直接使用到堆栈上，造成缓冲区溢出。

危险函数漏洞分析(快)				
可疑	函数名	函数地址	格式	格式字符串
1	sprintf	0x0004ea68	0...	%s.listnum
1	sprintf	0x0004ead4	0...	%s.list%d
1	sprintf	0x0004eb4c	0...	%s.list%d
1	sprintf	0x0004ebd8	0...	%s.list%d
1	sprintf	0x0004ec50	0...	%s.list%d
1	sprintf	0x0004ec98	0...	%s.listnum
1	sprintf	0x0004ecec	0...	%s.list%d
1	sprintf	0x0004ed64	0...	%s.list%d

```
[info] 从0x0004ebd8回溯字符串R2
[info] 回溯R11      0x0004ebd0      LDR      R2, [R11,#var_168]
[info] 回溯R11      0x0004ebc4      SUB      R2, R11, #-s
[info] 回溯R11      0x0004eba8      SUB      R3, R11, #-s
.....|
[info] 回溯R11      0x0004e9ec      STRB     R3, [R11,#c]
[info] 回溯R11      0x0004e9e4      STR      R1, [R11,#c+1]
[info] 回溯R11      0x0004e9e0      STR      R0, [R11,#var_168]
[info] 取消回溯SP   0x0004e9d0      ADD      R11, SP, #8
```


CVE-2020-13392

```
}  
}  
v17 = (char *)get_param(v2, (int)"funcname", (int)&unk_DDEE8);  
if ( *v17 )  
{  
    if ( !strcmp(v17, "save_list_data") )  
    {  
        v16 = get_param(v2, (int)"funcpara1", (int)&unk_DDEE8);  
        v15 = (char *)get_param(v2, (int)"funcpara2", (int)&unk_DDEE8);  
        sub_4E9CC((int)v16, v15, 0x7Eu);  
    }  
    else if ( !strcmp(v17, "LoadDhcpService") )  
    {  
        .  
    }  
}
```

```
1 int __fastcall sub_4E9CC(int a1, char *a2, unsigned __int8 a3)  
2 {  
3     int result; // r0  
4     unsigned __int8 c; // [sp+7h] [bp-16Dh]  
5     char *c_1; // [sp+8h] [bp-16Ch]  
6     int v6; // [sp+Ch] [bp-168h]  
7     char v7; // [sp+14h] [bp-160h]  
8     char v8; // [sp+1Ch] [bp-158h]  
9     char s; // [sp+11Ch] [bp-58h]  
10    char *v10; // [sp+15Ch] [bp-18h]  
11    int v11; // [sp+160h] [bp-14h]  
12    char *v12; // [sp+164h] [bp-10h]  
13  
14    v6 = a1;  
15    c_1 = a2;  
16    c = a3;  
17    memset(&s, 0, 0x40u);  
18    memset(&v8, 0, 0x100u);  
19    v11 = 0;  
20    if ( strlen(c_1) > 4 )  
21    {  
22        ++v11;  
23        v12 = c_1;  
24        while ( 1 )  
25        {  
26            v10 = strchr(v12, c);  
27            if ( !v10 )  
28                break;  
29            *v10++ = 0;  
30            memset(&s, 0, 0x40u);  
31            sprintf(&s, "%s.list%d", v6, v11);  
32            SetValue(&s, v12);  
33            v12 = v10;  
34            ++v11;  
35        }  
36    }  
37    return result;  
}
```

CVE-2020-13391

- 路由器的Web服务器httpd中存在一个缓冲区溢出漏洞。在处理POST请求的/goform/SetSpeedduizspeed_dir参数时，在sprintf中直接将值用于放置在堆栈上，造成缓冲区溢出。

```
43 v21 = (char *)sub_2B9D4(a1, (int)"speed_dir", (int)"0");
44 v20 = (char *)sub_2B9D4(v2, (int)"ucloud_enable", (int)"0");
45 v19 = sub_2B9D4(v2, (int)"password", (int)"0");
46 GetValue("speedtest.flag", &nptr);
47 if ( atoi((const char *)&nptr) )
48 {
49     v22 = 1;
50 }
51 else
52 {
53     SetValue("speedtest.flag", "1");
54     if ( atoi(v21) )
55     {
56         if ( !atoi(v20) )
57         {
58             SetValue("ucloud.en", "1");
59             SetValue("ucloud.syncserver", "1");
60             SetValue("ucloud.password", v19);
61             SetValue("qos.ucloud.flag", "1");
62             doSystemCmd((int)"cfm Post ucloud 0");
63         }
64         SetValue("speedtest.ret", "2");
65         doSystemCmd((int)"/bin/speedtest %d %d &");
66     }
67     else
68     {
69         SetValue("speedtest.ret", "4");
70         doSystemCmd((int)"cfm Post ucloud 5");
71     }
72 }
73 sprintf((char *)&s, "{\\"errCode\\":%d,\\"speed_dir\\":%s}", v22, v21);
74 return sub_9C9F0(v2, &s);
75 }
```

自动化IDA插件

- 改进方向：
 - 完善参数回溯逻辑，支持更复杂的指令语义识别，支持函数间分析
 - 完善漏洞判断逻辑，降低误报率
 - 加入动态污点分析作为辅助
 - 支持更多体系架构，如x86、MIPS等