

还没尝试用英语写过文章，试试我的语法，提前申明，THIS article will only be used in acadameic usage.

The Fortigate you might not know

From China Moew Moew Moew Security Research Team

Hello, you. I am a blue team security researcher from moew moew moew . Recently I realize that the network device is a good path for CTF pwn player to real vulnerability researcher. The network device security analysis is not a hard way like browser or CPU hardware to get a binary security job. And as the rising conflict of the world, keeping your assests secure is getting more and more complicated.

The FortiGate Next-Generation Firewall (NGFW), part of the Fortinet Security Fabric, is a robust firewall solution designed to protect organizations from both internal and external threats. It has a large group of customer with varaieties of good aspect: FortiGate NGFW provides automated protection against attacks, malware, and other [vulnerabilities](#). It offers end-to-end security across your entire network, including internal segmentation, perimeter, cloud, data center, distributed, and small business deployments.

FortiGate NGFW is known for its unparalleled AI-powered security performance, But i never used it. threat intelligence, and full visibility. It converges networking and security into a single operating system, making it easy to manage. With features like SD-WAN, switching, wireless, and 5G integration, FortiGate NGFW ensures robust protection for various use cases.

Getting firmware

After register a fortigate account, download Fortigate image rom fortigate download website, click SERVICE and then it will pop a page like the following.



ASSETS & ACCOUNTS



Asset Management



IAM

CLOUD MANAGEMENT



FortiClient EMS Cloud



FortiGate Cloud



FortiAnalyzer Cloud



FortiManager Cloud



FortiLAN Cloud



FortiSOAR Cloud



FortiSIEM Cloud



FortiExtender Cloud

CLOUD SERVICES



FortiSASE



FortiMail



FortiMonitor



SOCaaS



FortiRecon



FortiWeb Cloud



FortiConverter



FortiZTP



FortiSandbox Cloud



FortiCASB



Managed FortiGate



FortiToken Cloud



FortiCNP



FortiPhish



FortiGSLB



FortiVoice



FortiTrustID



Overlay as a Service



FortiGate CNF



FortiDAST



FortiDevSec



FortiPresence



FortiCamera Cloud



FortiCare Elite (Cloud)



FortiIPAM



FortiInsight



FortiDeceptor DaaS Cloud



FortiABP

Then click Asset Management , The firmware download button like this:



DOWNLOADS

[Firmware Download](#)
[VM Images](#)
[Service Updates](#)
[HQIP Images](#)
[Firmware Image Checksum](#)

RESOURCES

[Fortinet Support Community](#)
[Fortinet Video Library](#)
[FortiGuard Labs](#)
[Guidelines and Policies](#)
[Training](#)

Download image with fgt prefix and after that, three parameters should be configured in order to run fortigate correctly.

1. default gateway
2. ip and subnet mask of port1
3. DNS primary and secondary

In order to activate fortigate to use at most functions. You must configure the DNS. I prefer to set gateway to NAT.

Extremely easiest way to debug-environment deployment after FGT-7.4.0

c01c114317a6681fa7b31a97e0f187f7

Before you should know first:

1. GDB use command: `dump` to dump memory in to host machine, use command: `restore` to do the reverse——put a file into guest memory
2. DebugStub is a vmware feature that can trace the kernel after bootloader executed.
3. Busybox is a software that include almost common command of linux.such as `ls` , `ps` , `cp` ... Always been used in the embedded system.

Then there's a idea comes from my mind.What about write a shellcode to download busybox, download

So what about write shellcode to get a busybox

```
debugStub.listen.guest64 = "TRUE"
debugStub.listen.guest64.remote = "TRUE"
debugStub.port.guest64 = "4321"
monitor.debugOnStartGuest64 = "TRUE"
debugStub.hideBreakpoints = "TRUE"
```

1. get flatkc
2. get decrypted rootfs.gz

in 7.4.4, This is decrypted function,use "debugstub" to control rip to that xwrite, dump `rsi[rdx]`

```

_int64 sub_FFFFFFFF8170D56B()
{
    int v0; // esi
    __int64 v1; // rax
    int v2; // edx
    int v3; // ecx
    int v4; // r8d
    int v5; // r9d
    __int64 v6; // r12
    int v7; // edx
    int v8; // ecx
    int v9; // r8d
    int v10; // r9d
    int v11; // eax
    unsigned int v12; // ebx
    __int64 v13; // rax
    int v14; // ecx
    int v15; // r8d
    int v16; // r9d

    v0 = (int)off_FFFFFFFF817DD3D8;
    v1 = sub_FFFFFFFF8170D2E8(a070701000002d1, off_FFFFFFFF817DD3D8);
    if ( v1 )
        panic((unsigned int)&aS_24[1], v1, v2, v3, v4, v5);
    if ( qword_FFFFFFFF8183B080 )
    {
        printk((unsigned int)&unk_FFFFFFFF813D2010, v0, v2, v3, v4, v5);
        v6 = sub_FFFFFFFF8170D2E8(qword_FFFFFFFF8183B080, qword_FFFFFFFF8183B078 - qword_FFFFFFFF8183B080);
        if ( !v6 )
        {
            LABEL_9:
                free_initrd();
                goto LABEL_10;
        }
        sub_FFFFFFFF8170CF83();
        sub_FFFFFFFF8170D2E8(a070701000002d1, off_FFFFFFFF817DD3D8);
        printk((unsigned int)&unk_FFFFFFFF813D2048, v6, v7, v8, v9, v10);
        v11 = do_sys_open(4294967196LL, aInitrdImage, 32833LL, 448LL);
        v12 = v11;
        if ( v11 >= 0 )
        {
            v13 = xwrite((unsigned int)v11, qword_FFFFFFFF8183B080, qword_FFFFFFFF8183B078 - qword_FFFFFFFF8183B080);
            if ( qword_FFFFFFFF8183B078 - qword_FFFFFFFF8183B080 != v13 )
                printk((unsigned int)&unk_FFFFFFFF813D2088, v13, qword_FFFFFFFF8183B078 - qword_FFFFFFFF8183B080, v14, v15, v16);
            _close_fd((__QWORD *))(__readgsqword((unsigned int)&off_14D80) + 1576), v12);
            goto LABEL_9;
        }
    }
    LABEL_10:
        flush_delayed_fput();
        load_default_modules();
        return 0LL;
}

```

you could get a unpacked rootfs.gz.

Force to set \$rip into do_sys_open function.

```

Breakpoint 3, 0xffffffff8170d631 in ?? ()
1: x/30i $rip
=> 0xffffffff8170d631: callq 0xffffffff8170cf26
0xffffffff8170d636: mov %rax,%rsi
0xffffffff8170d639: mov 0x12da38(%rip),%rdx # 0xffffffff8183b078
0xffffffff8170d640: sub 0x12da39(%rip),%rdx # 0xffffffff8183b080
0xffffffff8170d647: cmp %rax,%rdx
0xffffffff8170d64a: je 0xffffffff8170d658
0xffffffff8170d64c: mov $0xffffffff813d2088,%rdi
0xffffffff8170d653: callq 0xffffffff803405f8
0xffffffff8170d658: mov %gs:0x14d80,%rax
0xffffffff8170d661: mov 0x628(%rax),%rdi
0xffffffff8170d668: mov %ebx,%esi
0xffffffff8170d66a: callq 0xffffffff804688b2
0xffffffff8170d66f: callq 0xffffffff8170d102
0xffffffff8170d674: callq 0xffffffff8044ba95
0xffffffff8170d679: callq 0xffffffff8170aa2a
0xffffffff8170d67e: xor %eax,%eax
0xffffffff8170d680: pop %rbx
0xffffffff8170d681: pop %r12
0xffffffff8170d683: pop %rbp
0xffffffff8170d684: retq
0xffffffff8170d685: push %rbp
0xffffffff8170d686: mov %rsp,%rbp
0xffffffff8170d689: push %r13
0xffffffff8170d68b: push %r12
0xffffffff8170d68d: push %rbx
0xffffffff8170d68e: movl $0x3,0x88d4c(%rip) # 0xffffffff817963e4
0xffffffff8170d698: movl $0x7,0x88d3e(%rip) # 0xffffffff817963e0
0xffffffff8170d6a2: mov 0x88d07(%rip),%r12 # 0xffffffff817963b0
0xffffffff8170d6a9: mov $0xffffffff813dlf8b,%rsi
0xffffffff8170d6b0: mov %r12,%rdi
(gdb) i r rsi
rsi 0xffff88807ba24000 -131389565288448
(gdb) x/s $rsi
0xffff88807ba24000: "\037\213\b"
(gdb) x/100xb $rsi
0xffff88807ba24000: 0x1f 0x8b 0x08 0x00 0x55 0x9c 0x43 0x66
0xffff88807ba24008: 0x00 0x03 0xa4 0xd7 0x63 0x8c 0x2e 0xc0
0xffff88807ba24010: 0xf3 0x20 0xea 0xb1 0x6d 0xdb 0xd6 0x19
0xffff88807ba24018: 0xdb 0xb6 0x6d 0x1b 0xef 0xd8 0xb6 0x7d
0xffff88807ba24020: 0xc6 0xb6 0x6d 0xdb 0xf6 0x9c 0xb1 0x6d
0xffff88807ba24028: 0xee 0x2f 0xd9 0x2c 0xee 0xe6 0xfe 0xb3
0xffff88807ba24030: 0xc9 0xbd 0xf5 0xa9 0x92 0x4e 0xd5 0xd3
0xffff88807ba24038: 0xfd 0xa1 0x2b 0xdd 0x4c 0x1c 0x4c 0x1c
0xffff88807ba24040: 0x4c 0xcc 0x4c 0xcc 0x7f 0x58 0xfe 0x18
0xffff88807ba24048: 0x1b 0x31 0x33 0xfd 0x27 0xd8 0x98 0xcd
0xffff88807ba24050: 0x4c 0x99 0xfe 0xdf 0xc3 0x94 0x9d 0x9d
0xffff88807ba24058: 0x8d 0xc5 0x88 0x8b 0xf3 0xbf 0x58 0x66
0xffff88807ba24060: 0xff 0xaf 0xea 0x58
(gdb) i r rdx
rdx 0x44bb28b 72069771
(gdb) i r rip
rip 0xffffffff8170d631 0xffffffff8170d631
(gdb)

```

then run this command to dump the decrypted rootfs.gz

```

(gdb) i r rip
rip 0xffffffff8170d631 0xffffffff8170d631
(gdb) dump binary memory rootfs744.gz $rsi $rdx
Invalid memory address range (start >= end).
(gdb) dump binary memory rootfs744.gz $rsi $rsi+$rdx

```

Good job. after few minutes to wait, we just get a holy decrypted rootfs.gz, which can be recognised as gzip compressed file.

```
root@ [REDACTED] /home [REDACTED] rti744# ls
flatkc rootfs744.gz rootfs.gz
root@ [REDACTED] /home/[REDACTED] rti744# file rootfs744.gz
rootfs744.gz: gzip compressed data, last modified: Tue May 14 17:16:05 2024, from Unix
root@ [REDACTED] /home [REDACTED] rti744#
```

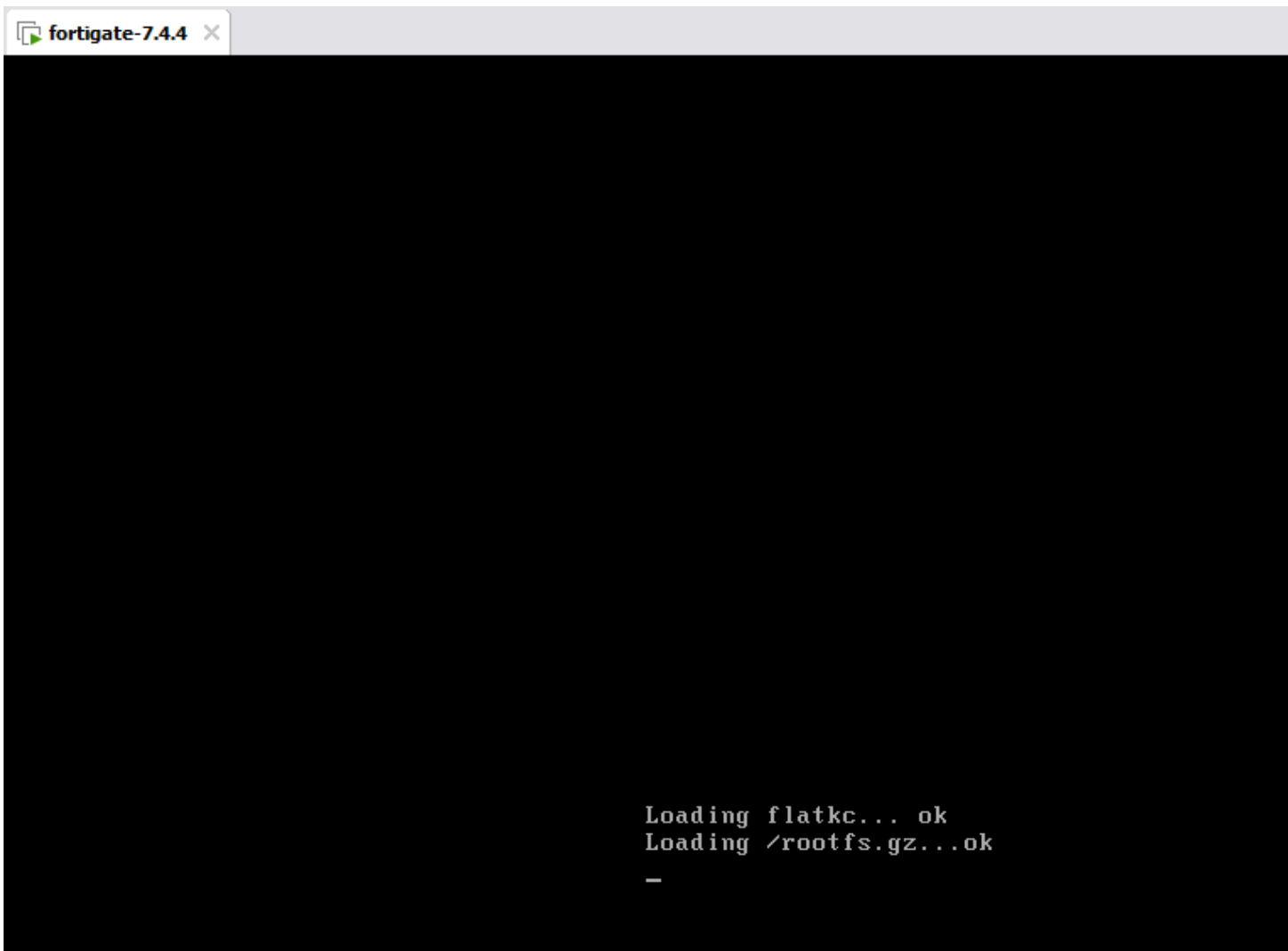
then use gunzip to decompress this file, and use `cpio -idmv < ./rootfs744` to get every single files

```
bin.tar.xz boot data data2 dev etc fortidev init lib lib64 migadmin.tar.xz node-scripts.tar.xz proc rootfs744 sbin sys tmp usr usr.tar.xz var
```

If your fortigate is in higher version, you can then execute `tar -xvf ./bin.tar.xz` to unpack bin. but in the lower version, fortigate gives sbin/ftar sbin/xz to de-xz and de-tar. This is already written by many articles, so i will not discuss lower version.

After boot the Fortigate, the vm machine will waiting for debugger to attach to the kernel because of debugstub.

By the way, before you attach to it, do not try to do anything stupid to the vm machine, do not poweroff or take a snapshot, or the vm machine will suspend until you restart VMX process and VMware process. (restart your computer is a easy way).



Then get your windows host ip, on the windows cmd panel, run `ipconfig` to get your host ip, such as [192.168.xxx.xxx](#) or [10.xxx.xxx.xxx](#). Attention, debugstub listend on the ip and port of your vmware host machine, not on the ip of fortigate. use gdb command `target remote 192.168.1.108:4321` to attach to debugstub

```
(gdb) target remote 192.168.1.108:4321
```

run `display/30i $rip` to show asm code.

```
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x00000000000100200 in ?? ()
1: x/30i $rip
=> 0x100200:    xor     %eax,%eax
    0x100202:    mov     %eax,%ds
    0x100204:    mov     %eax,%es
    0x100206:    mov     %eax,%ss
    0x100208:    mov     %eax,%fs
    0x10020a:    mov     %eax,%gs
    0x10020c:    mov     $0x200000,%rbp
    0x100213:    mov     0x260(%rsi),%ebx
    0x100219:    sub     $0x71b000,%ebx
    0x10021f:    add     %rbp,%rbx
    0x100222:    lea     0x711980(%rbx),%rsp
    0x100229:    xor     %rax,%rax
    0x10022c:    callq   0x100231
    0x100231:    pop     %rdi
    0x100232:    sub     $0x231,%rdi
    0x100239:    callq   0x7f48a3
    0x10023e:    lea     0x6fd54b(%rip),%rax        # 0x7fd790
    0x100245:    mov     %rax,0x6fd536(%rip)        # 0x7fd782
    0x10024c:    lgdt    0x6fd52d(%rip)            # 0x7fd780
    0x100253:    push    %rsi
    0x100254:    mov     %rsi,%rdi
    0x100257:    callq   0x7f85f0
    0x10025c:    pop     %rsi
    0x10025d:    mov     %rax,%rcx
    0x100260:    lea     0xc(%rip),%rdi            # 0x100273
    0x100267:    pushq   $0x8
    0x100269:    lea     0x1000(%rax),%rax
    0x100270:    push    %rax
    0x100271:    lretq
    0x100273:    lea     0x711980(%rbx),%rsp
(gdb) s
```

This is code is in decompressed `flatk` , which is implemented in `/boot` directory.

Remember our target is to find an easists way to implement a debug environment in FGT newest version.

I think use Vmware debugstub and try to write shellcode to get to it.

So our solution is :

0. do something to bypass security check.
1. find a address to hook
2. alloc a memory
3. use gdb `restore` command to put busybox in memory.

4. write the busybox in /bin and chmod 777
5. use gdb restore command to put smartctl in memory.
6. write the backdoor to smartctl and chmod 777
7. run diagnose hardware smartctl to execute busybox shell.

Let me explain above:

0. security check is important, i will talk later.
1. The hook address i choose is ping . because ping code is less impact on Fortigate system runtime.

the address is 0x299DA68 in /bin/init.

```
.text:00000000299DA68 loc_299DA68: ; CODE XREF: sub_299DA20+1C↑j
.text:00000000299DA68 mov edi, offset aSPingStatistic ; "\n--- %s ping statistics ---\n"
.text:00000000299DA6D xor eax, eax
.text:00000000299DA6F call _printf
.text:00000000299DA74 mov esi, cs:dword_C310B48
.text:00000000299DA7A mov edi, offset aDPacketsTransm ; "%d packets transmitted, "
.text:00000000299DA7F xor eax, eax
.text:00000000299DA81 call _printf
.text:00000000299DA86 mov esi, cs:dword_C310B4C
.text:00000000299DA8C mov edi, offset aDPacketsReceiv_1 ; "%d packets received, "
.text:00000000299DA91 xor eax, eax
.text:00000000299DA93 call _printf
```

2. when hooked success, use mmap syscall to alloc a big big big big big big big big memory, a little bit larger than busybox file.
3. memset the memory to 0 (In kernel if you mmap a the page will not be allocated instantly then we restore file in memory will fail

resources here

backdoor smartctl:

```
#/bin/busybox ash
/bin/busybox ash
```

hooked shellcode

only change memset addr is enough.

```

[BITS 64]
sub rsp, 8
call delete
call alloc_mem
mov r13, rax
mov rdi, r13
mov rsi, 0
mov rdx, 0x71fb38
mov rax, 0x43B3B0 ;memset
call rax

lea rdi, [rel busybox] ; 文件名指针
mov r15, 0xed96a ; len
nop
call write_file

lea rdi, [rel smartctl] ; smartctl
mov r15, 0x25 ; len
nop
call write_file

lea rdi, [rel gdb]
mov r15, 0x3f927f
nop
call write_file

lea rdi, [rel busybox] ; 第一个参数: 命令字符串
xor rsi, rsi
xor rdx, rdx
mov rax, 59 ; execve系统调用的系统调用号是59
syscall
ret

write_file:
    mov r10, rdi
    mov rax, 2 ; 系统调用号 (sys_open)
    mov rsi, 0x41 ; 标志
    mov rdx, 777 ; 权限 (rw-r--r--)
    syscall ; 调用内核
    mov r14, rax

```

```

mov rdi, rax
mov rax, 1                ; 系统调用号 (sys_write)
mov rsi, r13
mov rdx, r15              ; len
syscall                  ; 调用内核

; 关闭文件
mov rax, 3                ; 系统调用号 (sys_close)
mov rdi, r14              ; 文件描述符
syscall                  ; 调用内核

; chmod
mov rax, 90               ; syscall number for chmod
mov rdi, r10              ; pointer to the filename
mov rsi, 0777             ; set permissions to 777 (octal)
syscall                  ; make the system call
ret

```

```

father:
add rsp, 8
ret

```

```

;readfile:
;    mov rax, 2            ; 系统调用号 (sys_open)
;    lea rdi, [rel filename] ; 文件名指针
;    mov rsi, 0x0          ; 标志
;    mov rdx, 0x1ff        ; 权限 (rw-r--r--)
;    syscall              ; 调用内核
;    mov r14, rax
;
;    nop
;
;    mov rdi, rax
;    mov rax, 1            ; 系统调用号 (sys_write)
;    mov rsi, r13
;    mov rdx, 0x100        ; 消息长度
;    syscall              ; 调用内核
;
;    nop

;    mov rax, 3            ; 系统调用号 (sys_close)
;    mov rdi, r14          ; 文件描述符

```

```
;    syscall
```

```
alloc_mem:
```

```
    mov rax, 9
    mov rdi, 0x80000000
    mov rsi, 0x71fb38
    mov rdx, 7
    mov rcx, 0x22
    mov r10, 0x22
    mov r8, -1
    mov r9, 0
    syscall
    ret
```

```
delete:
```

```
; 设置系统调用为unlink（系统调用号为87）
mov rax, 87
; 设置第一个参数为文件名的地址
; 调用系统调用
syscall
ret
```

```
section .data
```

```
tmpaddr db 0,0,0,0,0,0,0,0
smartctl db '/bin/smartctl', 0      ; 文件名和路径
app db "ash", 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

busybox db '/bin/busybox',0        ; busybox字符串, 以null结尾
ash db 'ash',0                     ; ash字符串, 以null结尾
gdb db '/bin/gdb', 0
envlist dq 0                       ; 环境变量列表
```

1. gdb script: the \$func is hook addr we mentioned (hijack ping command at 0x299da68).

```
set {char [5]}(char *)0xFFFFFFFF80553104="\x31\xc0\xc3"  
# bypass fos_process_appraise
```

```
set $func = 0x2366158  
hbreak *$func  
command  
    restore 1.o binary $rip  
end  
# 1.o is hooked shellcode(above asm code compiled)
```

```
hbreak *($func+0x32)  
command  
    restore busybox binary $r13  
end
```

```
hbreak *($func+0x45)  
command  
    restore smartctl binary $r13  
end
```

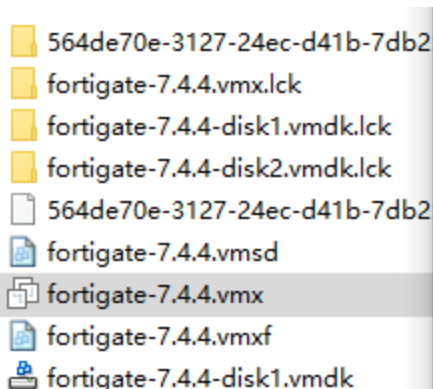
```
hbreak *($func+0x58)  
command  
    restore gdb binary $r13  
end
```

4. busybox:

download at your will and compile it.

after you prepared all resources, let's start:

1. write debugstub setting in your fortigate vmx file



```
debugStub.listen.guest64 = "TRUE"
debugStub.listen.guest64.remote = "TRUE"
debugStub.port.guest64 = "4444"
monitor.debugOnStartGuest64 = "TRUE"
debugStub.hideBreakpoints = "TRUE"

toolsInstallManager.updateCounter = "3"
tools.remindInstall = "FALSE"
```

2. turn on the fortigate vm machine
3. use gdb to attach to debugstub, ensure that your gdb is at resources directory, which has busybox smartctl (backdoor),
4. paste hooked shellcode to 1.asm, run command `nasm 1.asm -o 1.o` to get 1.o
5. press C continue to run
6. paste the gdb script to gdb

```
/Desktop/fortigate-utils-744# nasm 1.asm -o 1.o
```

```
(gdb) set {char [5]}(char *)0xFFFFFFFF805512EB="\x31\xc0\xc3"
(gdb)
(gdb) set $func = 0x299DA68
(gdb) hbreak *$func
Hardware assisted breakpoint 58 at 0x299da68
(gdb) command
Type commands for breakpoint(s) 58, one per line.
End with a line saying just "end".
>restore 1.o binary $rip
>end
(gdb)
(gdb) hbreak *($func+0x33)
Hardware assisted breakpoint 59 at 0x299da9b
(gdb) command
Type commands for breakpoint(s) 59, one per line.
End with a line saying just "end".
>restore busybox binary $r13
>end
(gdb)
(gdb) hbreak *($func+0x46)
Hardware assisted breakpoint 60 at 0x299daae
(gdb) command
Type commands for breakpoint(s) 60, one per line.
End with a line saying just "end".
>restore smartctl binary $r13
>end
```

if you want to change version
just modify 4 things:

1. In gdb script: modify addr of `fos_process_appraise`
2. In gdb script: modify hooked address (as I choose ping command in this example)
3. In hooked shellcode: modify addr of `memset`
4. In hooked shellcode: modify addr of

New safe feature in Fortigate

Since fortigate 7.4.0 has imported a new feature, some functions was added to improve the security performance. I didn't find any article to introduce this, only in official web page after my research has done.

<https://docs.fortinet.com/document/fortigate/7.4.0/new-features/249947/enhance-bios-level-signature-and-file-integrity-checking>

The main function is `fos_process_appraise` :

In 7.4.4, flatkc kernel decompressed, IDA hexray pseudo C code:

```

1 __int64 __fastcall fos_process_appraise_constprop_0(__int64 a1)
2
3 unsigned __int64 v1; // rax
4 int v2; // edx
5 int v3; // ecx
6 int v4; // r8d
7 int v5; // r9d
8
9 unsigned __int64 v6; // rbx
10 __int64 v7; // r14
11 int v8; // r13d
12
13 unsigned int i; // edx
14 __int64 result; // rax
15
16 int v11; // r13d
17 int v12; // r8d
18 int v13; // r9d
19 __int64 v14; // rax
20
21 int v15; // edx
22
23 int v16; // ecx
24
25 int v17; // r8d
26
27 int v18; // r9d
28
29 __int64 v19; // rax
30 __int64 v20; // rax
31
32 int v21; // [rsp+8h] [rbp-4B0h]
33 __int64 v22; // [rsp+10h] [rbp-4A8h]
34
35 int v23[11]; // [rsp+18h] [rbp-4A0h] BYREF
36
37 char v24; // [rsp+44h] [rbp-474h] BYREF
38
39 __int16 v25; // [rsp+45h] [rbp-473h]
40
41 char v26; // [rsp+47h] [rbp-471h]
42
43 __int64 v27[8]; // [rsp+48h] [rbp-470h] BYREF
44
45 char v28[512]; // [rsp+88h] [rbp-430h] BYREF
46
47 char v29[512]; // [rsp+288h] [rbp-230h] BYREF
48
49 unsigned __int64 v30; // [rsp+488h] [rbp-30h]
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

This function will be called when execute a binary or do sys_write syscall. If you try to:

1. write files in /bin /migadmin ... : syscall return -24.

```

.rodata:FFFFFFF81404F50 aBin          db '/bin',0
.rodata:FFFFFFF81404F55 aMigadmin    db '/migadmin',0
.rodata:FFFFFFF81404F5F aNodeScripts db '/node-scripts',0
.rodata:FFFFFFF81404F60 aSbin       db '/sbin',0
.rodata:FFFFFFF81404F73 aTools      db '/tools',0
.rodata:FFFFFFF81404F7A aUsr        db '/usr',0
.rodata:FFFFFFF81404F7F aLib64      db '/lib64',0
.rodata:FFFFFFF81404F86          align 8

```


2. execute binary outside will reboot immediately.

[I have no picture]

fortigate 7.4.4 flatkc decompress, kernel. hexray prrsudo code:

```

__int64 __fastcall fos_process_appraise_constprop_0(__int64 a1)
{
    unsigned __int64 v1; // rax
    int v2; // edx
    int v3; // ecx
    int v4; // r8d
    int v5; // r9d
    unsigned __int64 v6; // rbx
    __int64 v7; // r14
    int v8; // r13d
    unsigned int i; // edx
    __int64 result; // rax
    int v11; // r13d
    int v12; // r8d
    int v13; // r9d
    __int64 v14; // rax
    int v15; // edx
    int v16; // ecx
    int v17; // r8d
    int v18; // r9d
    __int64 v19; // rax
    __int64 v20; // rax
    int v21; // [rsp+8h] [rbp-4B0h]
    __int64 v22; // [rsp+10h] [rbp-4A8h]
    int v23[11]; // [rsp+18h] [rbp-4A0h] BYREF
    char v24; // [rsp+44h] [rbp-474h] BYREF
    __int16 v25; // [rsp+45h] [rbp-473h]
    char v26; // [rsp+47h] [rbp-471h]
    __int64 v27[8]; // [rsp+48h] [rbp-470h] BYREF
    char v28[512]; // [rsp+88h] [rbp-430h] BYREF
    char v29[512]; // [rsp+288h] [rbp-230h] BYREF
    unsigned __int64 v30; // [rsp+488h] [rbp-30h]

    v30 = __readgsqword(0x28u);
    v22 = *(_QWORD *)(a1 + 32);
    v1 = d_path(a1 + 16, v28, 511LL);
    v6 = v1;
    if ( v1 > 0xFFFFFFFFFFFFFFFF000LL )
    {
        printk((unsigned int)&unk_FFFFFFFF81405ED8, (unsigned int)v28, v2, v3, v4, v5);
        ((void (*)(__int64, __int64, __int64, __int64, _QWORD, const char *, ...))off_FFFFFFFF816BAI
            36864LL,
            255LL,

```

```

255LL,
20233LL,
0LL,
"severity=alert msg=\"[The length of executable filename is longer than 512](%s).\\",
*(const char **)(*(__QWORD *)(a1 + 24) + 40LL));
msleep(10000LL);
kernel_restart(0LL);
return (unsigned int)v6;
}
else
{
v7 = sub_FFFFFFFF8055107D(v1);
if ( v7 )
{
v8 = 3;
v24 = dword_FFFFFFFF8165F6D4;
for ( i = 0; ; i = 0 )
{
do
{
v27[i] = 0LL;
v27[i + 1] = 0LL;
v27[i + 2] = 0LL;
v27[i + 3] = 0LL;
i += 4;
}
while ( i < 8 );
result = ima_calc_file_hash(a1, &v24);
if ( !(_DWORD)result )
break;
if ( !--v8 )
{
if ( (int)result < 0 )
return result;
break;
}
}
v25 = 0;
v26 = 0;
v24 = dword_FFFFFFFF8165F6D4;
}
result = memcmp((__QWORD *)(v7 + 584) + 4LL, v27, *(unsigned __int8 *)((__QWORD *)(v7 + 584) + 4LL));
if ( (_DWORD)result )
{

```

```

ima_pr_emerg(3LL, v6);
sub_FFFFFFFF80551250(aNew, v27, *(unsigned __int8 *)((_QWORD *)(v7 + 584) + 1LL));
sub_FFFFFFFF80551250(aOld, *(_QWORD *)(v7 + 584) + 4LL, *(unsigned __int8 *)((_QWORD *)
printk((unsigned int)&unk_FFFFFFFF81405E83, *(_QWORD *)(v22 + 80), v15, v16, v17, v18);
time64_to_tm(*(_QWORD *)(v22 + 104), 0LL, v23);
v21 = v23[0];
printk(
    (unsigned int)&unk_FFFFFFFF81405F68,
    (unsigned int)aModifiedTime,
    v23[6] + 1900,
    v23[4] + 1,
    v23[3],
    v23[2]);
v19 = sub_FFFFFFFF80552372(3LL);
((void (__fastcall *))(__int64, __int64, __int64, __int64, _QWORD, __int64, unsigned __i
    36864LL,
    255LL,
    255LL,
    20234LL,
    0LL,
    v19,
    v6,
    v21);
msleep(5000LL);
kernel_restart(0LL);
return 4294967283LL;
}
}
else
{
    v11 = (int)off_FFFFFFFF8165F6C0[0];
    if ( !(unsigned int)strcmp(off_FFFFFFFF8165F6C0[0], v6)
        || (v11 = (int)off_FFFFFFFF8165F6C8, !(unsigned int)strcmp(off_FFFFFFFF8165F6C8, v6)) )
    {
        memset(v29, 0, sizeof(v29));
        snprintf((unsigned int)v29, 511, (unsigned int)aSX_0, v11, v12, v13);
        if ( (int)sub_FFFFFFFF80551F40(a1, v6, v29) < 0 )
        {
            ima_pr_emerg(0LL, v6);
            v20 = sub_FFFFFFFF80552372(0LL);
            ((void (__fastcall *))(__int64, __int64, __int64, __int64, _QWORD, __int64, unsigned __i
                36864LL,
                255LL,

```


d\FGT-7.4.2\vmlinux742.i64

0

The latest version I found this symbol is 7.4.2.

The latest version I didn't found this symbole is 7.4.4(latest)

The key to find this function is :

```
__int64 __fastcall fortism_file_open_part_0_cold(__int64 a1, __int64 a2, __int64 a3, __int64 a4, int a5, int a6)
{
    const char *v6; // r13

    printk((unsigned int)&unk_FFFFFFFF81401450, (unsigned int)aFortismFileOpe, 78, (_DWORD)v6, a5, a6);
    ((void (*)(__int64, __int64, __int64, __int64, _QWORD, const char *, ...))off_FFFFFFFF816B8888[0])(
        36864LL,
        255LL,
        255LL,
        20230LL,
        0LL,
        "msg=\"[Write Violation: try to write readonly file](%s).\"",
        v6);
    return 4294967283LL;
}
```

So we can found this function in 7.4.4

```

signed __int64 __fastcall sub_FFFFFFFF805503D3(__int64 a1)

unsigned __int64 result; // rax
const char *v2; // r13
_QWORD *v3; // rbx
__int64 v4; // rax
__int64 v5; // rbx
__int64 v6; // rax
int v7; // r8d
int v8; // r9d
char v9[496]; // [rsp+270h] [rbp-220h] BYREF
unsigned __int64 v10; // [rsp+470h] [rbp-20h]

v10 = __readgsqword(0x28u);
result = d_path(a1 + 16, v9, 511LL);
v2 = (const char *)result;
v3 = &unk_FFFFFFFF8165F640;
if ( result <= 0xFFFFFFFFFFFFFFFF000LL )
{
    while ( 1 )
    {
        v4 = strlen(*v3);
        if ( !(unsigned int)strncmp(v2, *v3, v4) )
            break;
        if ( ++v3 == (_QWORD *)&unk_FFFFFFFF8165F698 )
            return 0LL;
    }
    v5 = 0LL;
    while ( 1 )
    {
        v6 = strlen((&off_FFFFFFFF8165F5E0)[v5]);
        result = strncmp(v2, (&off_FFFFFFFF8165F5E0)[v5], v6);
        if ( !(_DWORD)result )
            break;
        if ( ++v5 == 5 )
        {
            printk((unsigned int)&unk_FFFFFFFF81405040, (unsigned int)aFortismFileOpe, 80, (_DWORD)v2, v7, v8);
            ((void (*)(__int64, __int64, __int64, __int64, _QWORD, const char *, ...))off_FFFFFFFF816BAD48[0])(
                36864LL,
                255LL,
                255LL,
                20230LL,
                0LL,
                "msg=\"[Write Violation: try to write readonly file](%s).\"",
                v2);
            return 4294967283LL;
        }
    }
}
return result;

```

When attempting to `sys_write` to a runtime directory (such as `/migadmin`), it will return an error and log this critical operation. However, the oversight is that it uses `dpath` to obtain the absolute path, which prevents symbolic link attacks. Yet, it didn't check the parent directory. As a result, you can rename `/migadmin` to `/test` and create a symbolic link from `/bin` to `/test` to bypass this check.

```

        break;
    if ( !--v19 )
    {
        v16 = d_path(a1 + 16, (__int64)v24, 511);
        if ( v16 > 0xFFFFFFFFFFFFFFFF000LL )
            ima_pr_emerg(4LL, 0LL);
        else
            ima_pr_emerg(5LL, v16);
        return v5;
    }
    v21 = 0;
    v22 = 0;
    v20 = c01c114317a6681fa7b31a97e0f187f7;
}

```

dpath的检查

debug enviroment conclusion

As I indroduced the new security policy in new fortigate, How to bypass it?

For fos_process_appraiase

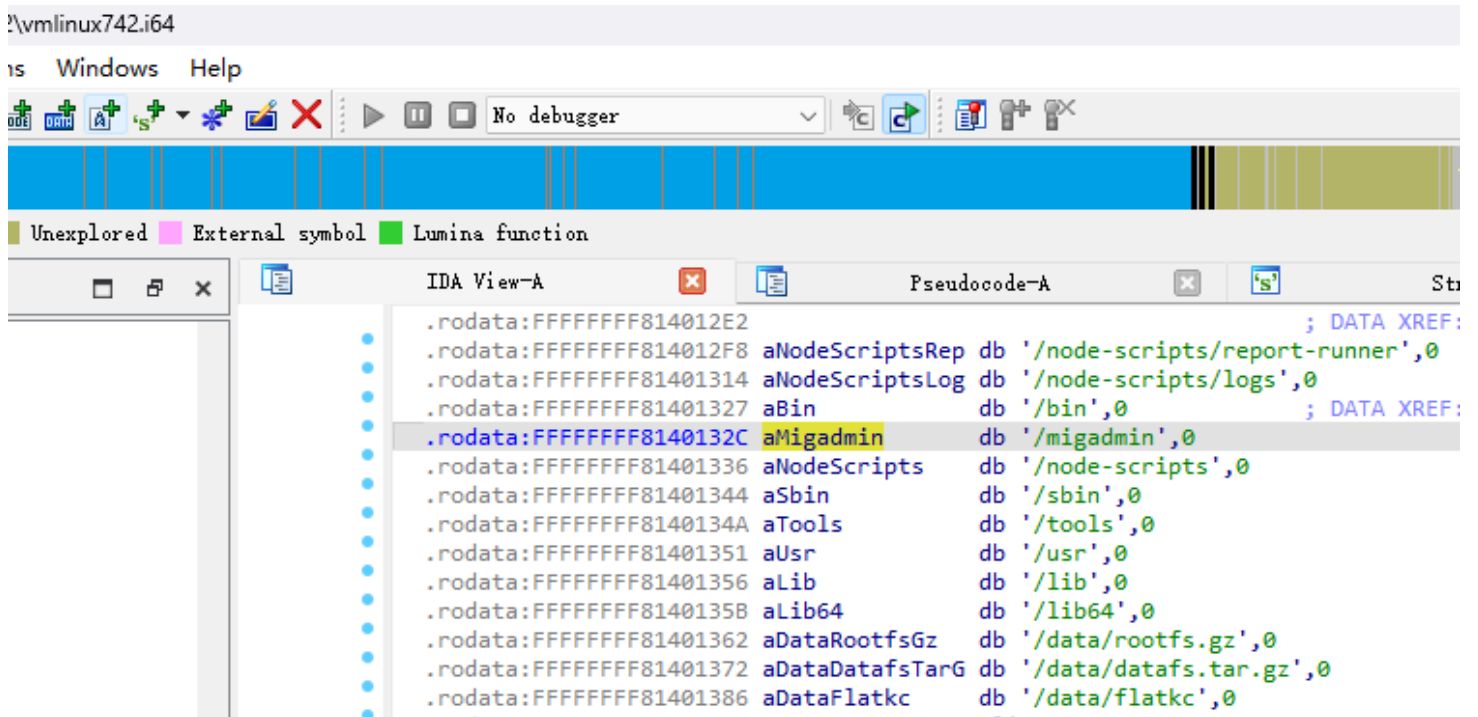
1. rename migadmin to another name such as testtest
2. create a link named migadmin to testtest
3. modify testtest.

So you can change web runtime enviroment

(can use this to Persistence exploit, not only for debug environment deploy)

For fos_fileopen

edit memory in vmlinux in debugstub, change string /bin to /ain to avoid check, change string /migadmin to /aaaa



After do that

find a addr to hook into shellcode, I choose this position

search string "packets transmitted"

```
.rodata:0000000039ABC29 ; const char aDPacketsTransm[]
.rodata:0000000039ABC29 aDPacketsTransm db '%d packets transmitted, ',0
.rodata:0000000039ABC29 ; DATA XREF: sub_28019B0+157
```

find this function

```
void sub_28019B0()
{
    if ( qword_FE498E0 )
    {
        if ( dword_FE499D0 )
            goto LABEL_3;
        printf("\n--- %s ping statistics ---\n", (const char *)qword_FE498E0);
        printf("%d packets transmitted, ", (unsigned int)dword_FE49900);
        printf("%d packets received, ", (unsigned int)dword_FE49904);
        if ( dword_FE49908 )
            printf("%d duplicates, ", (unsigned int)dword_FE49908);
        if ( dword_FE49900 )
            printf("%d%% packet loss\n", 100 * (dword_FE49900 - dword_FE49904) / (unsigned int)dword_FE49900);
        if ( SHIDWORD(qword_4B5E360) > 7 && dword_FE49904 )
            printf(
                "round-trip min/avg/max = %lu.%lu/%lu.%lu/%lu.%lu ms\n",
                qword_4B5E410 / 0xAuLL,
                qword_4B5E410 % 0xAuLL,
                qword_FE49920 / (unsigned __int64)(unsigned int)(dword_FE49908 + dword_FE49904) / 0xA,
                qword_FE49920 / (unsigned __int64)(unsigned int)(dword_FE49908 + dword_FE49904) % 0xA,
                qword_FE49918 / 0xAuLL,
                qword_FE49918 % 0xAuLL);
        fflush(stdout);
    }
    dword_FE499D0 = 1;
LABEL_3:
    dword_FE499C0 = 1;
    if ( dword_4B5E408 >= 0 )
        close(dword_4B5E408);
    dword_4B5E408 = -1;
}
```

set hbreak on this function in debugstub gdb

```
(gdb) hbreak *0x0000000028019B0
Hardware assisted breakpoint 3 at 0x28019b0
(gdb) c
Continuing.
```

in fortigate, use `exec ping 1.1.1.1` and `ctrl-c` to get into break

```
FortiGate-UM64 # exec ping 1.1.1.1
^C_
```

write gdb script into debugstub gdb

```
(gdb) set $func = 0x2366158
(gdb) hbreak *$func
Hardware assisted breakpoint 4 at 0x2366158
(gdb) command
Type commands for breakpoint(s) 4, one per line.
End with a line saying just "end".
>restore 1.o binary $rip
>end
(gdb)
(gdb)
(gdb)
(gdb) hbreak *($func+0x32)
Hardware assisted breakpoint 5 at 0x236618a
(gdb) command
Type commands for breakpoint(s) 5, one per line.
End with a line saying just "end".
>restore busybox binary $r13
>end
(gdb)
(gdb) hbreak *($func+0x45)
Hardware assisted breakpoint 6 at 0x236619d
(gdb) command
Type commands for breakpoint(s) 6, one per line.
End with a line saying just "end".
>restore smartctl binary $r13
>end
(gdb)
(gdb) hbreak *($func+0x58)
Hardware assisted breakpoint 7 at 0x23661b0
(gdb) command
Type commands for breakpoint(s) 7, one per line.
End with a line saying just "end".
>restore gdb binary $r13
>end
(gdb) █
```

press continue and you will get a debug environment if you run
diagnose hardware smartctl