Security Advisory

Vulnerability Title: Improper IOCTL Input Handling in FortKnox Personal Firewall 2015

Vendor	NETGATE / FortKnox
Product	FortKnox Personal Firewall 2015
Severity	High
Affected Version	Build 16.0.405.0

Impact

This vulnerability could allow a malicious attacker to gain Administrator privileges on a system from a user account.

Details

Fortknox Personal Firewall (build 16.0.405.0) is personal firewall solution that allows you to protect a PC against hacker attacks, Trojans, spyware and internet threats. Security issues discovered on Fortknox Personal Firewall. The vulnerability allow a low privileged user to execute code as SYSTEM by exploiting a vulnerability in the Fortknox Personal Firewall (fortknoxfw.sys) kernel mode driver. This is a 'trusted value vulnerability' that can be triggered through a specific IOCTL with a specifically crafted buffer, to force the driver to validate an improper IOCTL.

Vulnerable Device, "*Device**fortknoxfw_ctl*":

```
; vulnerable device
.text:00011365                push    offset aDeviceFortkn_0          ; "\\Device\\fortknoxfw_ctl"
.text:0001136A                lea     eax, [ebp+DestinationString]
.text:0001136D                push    eax                             ; DestinationString
.text:0001136E                call    esi                             ; RtlInitUnicodeString
```

Vulnerable IOCTL. Check if match:

```
; check if ioctl match in order to proceed
.text:0001600D                cmp     eax, 8E86200Ch          ; ioctl must 0x8E86200C
.text:00016012                jz      short loc_1604D         ; it will compare with eax, if not match, exit. else jump to checking
```

Attacker input send buffer:

```
; check input from sender, perform checking on the input, validate input on function sub_150E4
.text:0001604D loc_1604D:                                     ; CODE XREF: sub_15FC4+4D↑j
.text:0001604D                cmp     ecx, 98h                ; compare 0x98 with ECX, this should be send by attacker
.text:00016053                jnz     loc_16276               ; if send buffer match with 0x98, proceed without jump
.text:00016059                mov     eax, [ebp+arg_4]        ; arg_4 is our input
.text:0001605C                cmp     eax, ebx                ; compare eax with ebx
.text:0001605E                jz      loc_16276               ; if input (eax) zero value, it will jump to exit
.text:00016064                push    eax                     ; push the input (eax) into stack
.text:00016065                push    dword ptr [eax]
.text:00016067                call    sub_150E4               ; this call is basically do checking of input
```

Data validation:

```
; validation of input (data pre-processing)
; check #1
.text:0001518D loc_1518D:                              ; CODE XREF: sub_150E4+30↑j
.text:0001518D                 push    31565244h       ; Tag
.text:00015192                 push    98h             ; NumberOfBytes
.text:00015197                 push    edi             ; PoolType
.text:00015198                 call    ds:ExAllocatePoolWithTag ; allocates pool memory, our bytes here 0x98 = 152 bytes
.text:0001519E                 mov     ebx, eax        ; copy eax into ebx (buffer)
.text:000151A0                 cmp     ebx, edi        ; then compare edi with buffer (ebx)

; check #2
.text:000151A4                 mov     eax, [ebp+arg_4]        ; our buffer will copy into eax
.text:000151A7                 push    26h             ;
.text:000151A9                 mov     esi, eax        ;
.text:000151AB                 pop     ecx             ;
.text:000151AC                 mov     edi, ebx        ;
.text:000151AE                 rep movsd               ; memcpy(arg_4, a2, 0x98), here our buffer will copy

; check #3
.text:000151B0                 mov     eax, [eax+90h]  ;
.text:000151B6                 test    eax, eax        ;
.text:000151B8                 jz      short loc_151F6         ; check the length of buffer we send
.text:000151BA                 lea     esi, [eax+1]    ;
.text:000151BD loc_151BD:                              ; CODE XREF: sub_150E4+DB↑j
.text:000151BD                 mov     cl, [eax]       ; overflow trigger here due to invalid length checking on the input buffer
.text:000151BF                 inc     eax             ; overflow
.text:000151C0                 test    cl, cl          ; overflow
```

Crash Dump:

```
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

Use !analyze -v to get detailed debugging information.

BugCheck D1, {41414141, 2, 0, 883ab1bd}

Unable to load image \SystemRoot\system32\drivers\fortknoxfw.sys, Win32 error 0n2
*** WARNING: Unable to verify timestamp for fortknoxfw.sys
*** ERROR: Module load completed but symbols could not be loaded for fortknoxfw.sys
Probably caused by : fortknoxfw.sys ( fortknoxfw+51bd )

Followup: MachineOwner
---------

kd> !analyze -v
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

DRIVER_IRQL_NOT_LESS_OR_EQUAL (d1)
An attempt was made to access a pageable (or completely invalid) address at an
interrupt request level (IRQL) that is too high.  This is usually
caused by drivers using improper addresses.
If kernel debugger is available get stack backtrace.
Arguments:
```

```
Arg1: 41414141, memory referenced
Arg2: 00000002, IRQL
Arg3: 00000000, value 0 = read operation, 1 = write operation
Arg4: 883ab1bd, address which referenced memory

Debugging Details:
------------------


READ_ADDRESS: GetPointerFromAddress: unable to read from 82b70718
Unable to read MiSystemVaType memory at 82b501a0
 41414141

CURRENT_IRQL:  2

FAULTING_IP:
fortknoxfw+51bd
883ab1bd 8a08          mov     cl,byte ptr [eax]

DEFAULT_BUCKET_ID:  VISTA_DRIVER_FAULT

BUGCHECK_STR:  0xD1

PROCESS_NAME:  mop.exe

TRAP_FRAME:  942db870 -- (.trap 0xffffffff942db870)
ErrCode = 00000000
eax=41414141 ebx=86fd5b90 ecx=00000000 edx=00026b6d esi=41414142 edi=86fd5c28
eip=883ab1bd esp=942db8e4 ebp=942db8f4 iopl=0         nv up ei pl nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010206
fortknoxfw+0x51bd:
883ab1bd 8a08          mov     cl,byte ptr [eax]         ds:0023:41414141=??
Resetting default scope

LAST_CONTROL_TRANSFER:  from 883ab1bd to 82a475cb

STACK_TEXT:
942db870 883ab1bd badb0d00 00026b6d 82b278c0 nt!KiTrap0E+0x2cf
WARNING: Stack unwind information not available. Following frames may be wrong.
942db8f4 883ac06c 00000000 85199300 8b8b50d8 fortknoxfw+0x51bd
942db90c 883a7005 8e86200c 85199300 942db93c fortknoxfw+0x606c
942dbafc 82a3d593 8621ff08 8b8b50d8 8b8b50d8 fortknoxfw+0x1005
942dbb14 82c3199f 851b5ef0 8b8b50d8 8b8b5148 nt!IofCallDriver+0x63
942dbb34 82c34b71 8621ff08 851b5ef0 00000000 nt!IopSynchronousServiceTail+0x1f8
942dbbd0 82c7b3f4 8621ff08 8b8b50d8 00000000 nt!IopXxxControlFile+0x6aa
942dbc04 82a441ea 000000e0 00000000 00000000 nt!NtDeviceIoControlFile+0x2a
942dbc04 777270b4 000000e0 00000000 00000000 nt!KiFastCallEntry+0x12a
0028fdd8 00000000 00000000 00000000 00000000 0x777270b4
```

```
STACK_COMMAND:  kb

FOLLOWUP_IP:
fortknoxfw+51bd
883ab1bd 8a08          mov     cl,byte ptr [eax]

SYMBOL_STACK_INDEX:  1

SYMBOL_NAME:  fortknoxfw+51bd

FOLLOWUP_NAME:  MachineOwner

MODULE_NAME: fortknoxfw

IMAGE_NAME:  fortknoxfw.sys

DEBUG_FLR_IMAGE_TIMESTAMP:  549983c7

FAILURE_BUCKET_ID:  0xD1_fortknoxfw+51bd

BUCKET_ID:  0xD1_fortknoxfw+51bd

Followup: MachineOwner
---------
```

Report (Responsible Disclosure)

Jan 29, 2016 – Asking for security team e-mail to report issue.
Jan 29, 2016 – Someone replying asked to write to him.
Jan 29, 2016 – Sending report of vulnerability information.
Jan 30, 2016 – Asking for update from vendor.
Jan 30, 2016 – Vendor replied, so far only BSOD.
Feb 1, 2016 – Vendor asked for POC.
Feb 1, 2016 – Vendor still didn't understand how the BSOD works and why is it happened like that.
Feb 2, 2016 – Asking for update from vendor.
Feb 2, 2016 – Advise vendor to fix memory handling issue.
Feb 2, 2016 – Vendor will fix in the next version of update.
Feb 3, 2016 – Try to ask for CVE, no more updates from vendor.