Click to prove you're human

========== you can’t perform that action at this time. The creation of a driver for 64-bit Windows to interact with Dongles is currently not possible through Safenet Inc. However, the author has developed an alternative solution to enable interaction between Dongles and 64-bit NTVDMx64. To accomplish this, the author created a new driver and VDD component that allows interaction with the dongle on 64-bit NTVDMx64. This is done in the form of a driver, but it can also be achieved through emulation using UCLHASP functions for simple HASP dongles. The first step to resolving this issue is to check if your DOS application uses the INT 6 (Invalid opcode) services for decryption of crypted code. If not, you can use the Dongle usermode emulation as an alternative solution, which does not require installing a driver on your 64-bit machine and avoids potential instability issues. However, if this approach does not work, you may want to consider using the full driver variant or the UCLHASP emulator for best stability. The latter is also compatible with most MS-DOS emulations, including NTVDM. To use the UCLHASP emulator, you need to download the uclhaspemu.zip and unpack it to a directory. Then, run install.cmd to install the VDD and the emulator library. You should add your emulator dongle dump .reg file to registry keys in specific paths. Alternatively, you can also use the UCLHASP emulator as a plain DOS device driver to enable interaction with Dongles on bare-metal MS-DOS installations or in emulators like DOSBOX-X. This solution is compatible with most MS-DOS emulations but requires a working DOS emulator with support for device drivers reading full chunks of memory. The UCLHAPS.SYS driver takes an unencrypted Unidump-File as parameter, which can be created using h5dmp and fed to UniDump2Reg to create a .reg file containing the plain binary dongle dump.UCLHASP.SYS file download and configuration for NTVDM and DOS applications ========== Download the UCLHASP.SYS file from "Releases" to a location on your system that can be easily loaded later. Then, dump your Unidump file (e.g., HASP.DMP) to a directory of your DOS machine. Dump your Unidump file to a directory of your DOS-machine, so that you can pass its path and name as a parameter when loading the UCLHASP.SYS driver. Make your DOS or DOS emulation load the UCLHASP.SYS on CONFIG.SYS with DEVICE= command. Under plain DOS, this line has to go into CONFIG.SYS, under NTVDM into CONFIG.NT and under DosBox-X into the section named [config] in dosbox-x.conf. As mentioned above, Unidump file's name is assumed as HASP.DMP: DEVICE=UCLHASP.SYS HASP.DMP The DEVICE= statement should be at the bottom of your file/section so that the chance for interfering with DOS Version-checks for other drivers that may get loaded isn't high. The HARDLOCK.SYS driver is responsible for communicating with a hardware dongle. The end-user application links to the HASP library, which interfaces either directly with the dongle or through the HASPDOS.SYS driver on Windows NT due to lack of direct port access. Detection of NTVDM occurs via system call for DOS version, and if it's 5.50 (BX=3205h), the HASPDOSDRV driver is called. The HASP library uses a special buffer structure, described in haspintl.h, which stores register contents from the caller. The reply is received in this buffer. The table below outlines basic DOS HASP functions: | SERVICE | CALL | RETURN | | BH=FUNC, BL=PORT | | AX : return codes | | +----------+----------+----------+----------+ |1. | ISHASP | AX: 0 - HASP NOT FOUND, 1 - HASP FOUND | | +----------+----------+----------+----------+ |2. | HASPCODE | CX: password 1, DX: password 2 | BX: return codes | | +----------+----------+----------+ |3. | READMEMO | CX: status, DI: memory address, SI: data | | | +----------+----------+----------+ |4. | WRITEMEMO | CX: status, DI: memory address, SI: data | | | +----------+----------+----------+ |5. | HASPSTATUS | AX: memory size, BX: HASP type, CX: LPT_NUM | | | +----------+----------+----------+ |6. | HASPID | AX: IDLOW, BX: IDHIGH, CX: status | | The DOS device driver must be loaded via CONFIG.NT file (NTVDM variant of CONFIG.SYS) through the DEVICE= statement. The driver flags indicate a character device driver named HASPDOSDRV. The initialization routine registers an INT 2f Multiplexter Interrupt handler for function 5000h and returns magic numbers in AX-DX. It also uses handlers for OPEN, CLOSE, and INPUT (read). On OPEN, it registers with the HASPVDD.DLL via BOP interface, on CLOSE unregisters itself, and on INPUT passes the buffer to VDD. In the HASPDOS driver, there's an unused routine called CALLNEWVDDHASP, resembling the interface seen later in HASPNT.SYS. However, this is irrelevant as it cannot be called from the VDD. The dispatch routine gets a pointer and count as parameters and calls its internal CALLVDDHASP routine with these parameters. This routine decodes an "encrypted" buffer passed to it using an encryption scheme.the HASPDOS.SYS driver communicates with the upper-level drivers through three different implementations. first, there's the reimplemented HASPDOS.SYS driver that handles communications with the HASPVDD.DLL driver. second, there's the HASPEMU.SYS emulation driver which simply replays packets of call type #2 (HASPCODE) for extremely simple applications. lastly, there's the UCLHASP.SYS driver that emulates a MemoHasp in plain DOS given a UniDump file. the reimplemented HASPDOS.SYS driver opens a device handle to the downlevel HARDLOCK.SYS device node \Device\FNT0 and sends an initialization request IRP HASP_IOCTL_DOS_INITIALIZE to it. it then listens for irp_mj_read requests, decrypts the passed-in buffer via previously mentioned decryption routines, and converts the buffer to driver-internal format. this converted buffer structure is then passed down to the HARDLOCK.SYS driver via IOCTL HASP_IOCTL_DOS_DISPATCH. unfortunately, this ioctl cannot be accessed by user-mode applications, making it mandatory to implement a driver to access this ioctl code. the hardlock.sys driver is the core of the whole process and takes ioctls from callers to create requests to the dongle from passed-in buffer structures. multiple possible structures exist for different workflows, with the current implementation focusing on stability and shortest code paths. drivers here have a focus on stability and shortest code path. translation functions that are normally in haspnt.sys have been transferred to user-mode HASPVDD.DLL driver as it eliminates potential bugs and crashes that can cause bsods or hangs. ========== To get the HASP dongle working with DebugView, start by doing a normal setup with HASPDOS.SYS driver and then you can shortcut it using HASPEMU.SYS. First, set up the dongle via a normal setup with HASPVDD.DLL. Then use DebugView to see the communication between the dongle and your application. # Understanding the Communication When the dongle is connected, you will see the following output in DebugView: _____ [Ln] [Ticks] [Service] [Param1/AX] [Param2/BX] [Param3/CX] [Param4/DX] [Para1R/DI] [Para2R/SI] [Para3R/ES] [Para4R/AX] HASP OUT [28] AB 00 00 00 01 00 00 00 AD BA 00 00 00 00 00 00 EF BE 00 00 AD DE 00 00 01 00 00 00 01 00 00 00 00 00 00 00 AD BA 00 00 HASP IN [28] AB 00 00 00 01 00 01 00 01 00 00 00 01 00 00 00 00 00 00 00 AD BA 00 00 01 00 00 00 01 00 00 00 00 00 00 00 AD BA 00 00 The output can be interpreted using the table in haspintl.h. # Interpreting the Output Service: 1 Input parameters: Don't care Returns: AX = 1 - HASP FOUND, BX = 1 - Address of HASP dongle = 1, CX = 0 - Success The next call is of interest: _____ [Ln] [Ticks] [Service] [Param1/AX] [Param2/BX] [Param3/CX] [Param4/DX] [Para1R/DI] [Para2R/SI] [Para3R/ES] [Para4R/AX] HASP OUT [28] C8 02 00 00 02 00 00 00 AD BA 00 00 00 00 00 00 EF BE 00 00 AD DE 00 00 01 00 00 00 01 00 00 00 00 00 00 00 AD BA 00 00 HASP IN [28] C8 02 00 00 02 00 02 00 34 12 00 00 78 56 00 00 BC 9A 00 00 F0 DE 00 00 34 12 00 00 78 56 00 00 BC 9A 00 00 F0 DE 00 00 Interpreted: Service: 2 Input parameters: AX = BAAD = Seed code, CX = BEEF = Password 1, DX = DEAD = Password 2 Returns: AX = 1234 = Return code 1, BX = 5678 = Return code 2, CX = 9ABC = Return code 3, DX = DEF0 = Return code 4 # Loading the Driver To load the driver, use devload haspemu.sys 1234 5678 9abc def0. Alternatively, you can use DEVLOAD.COM in the haspdos directory to load it on demand. The UCLHASP driver was ported from 32-bit Windows to MS-DOS and requires modifications to work properly. The loader code reads a Unidump file with data to present to the calling application.the HASP_IOCTL_DOS_DISPATCH IOCTL code, similar to the underlying HARDLOCK.SYS driver's approach. However, due to this being an internal IOCTL, packets must be forwarded, which was previously mentioned. This interface resembles CALLNEWVDDHASP, where processing and preparation of the expected buffer are handled in usermode by the HASPVDD.DLL driver, preventing potential crashes from affecting the system stability. A notable advantage is that all necessary processing and preparation work is done in usermode, making the system less unstable in case of a crash. The call itself can be seen in CallHardlock function in hardlockdrv.c. **Note:** This interface is NOT compatible with the original HASPNT.SYS driver. It's mainly useful for logging purposes on 32-bit systems where you installed the HDD32 device driver 4.102.5.22 from Aladdin. With this mode, you can play man-in-the-middle and sniff calls to the underlying dongle by replacing the original HASPVDD.DLL driver with this one and setting specific registry keys. **Enabling and Configuring** To enable this interface and log call traffic between the application and the dongle, you need to: 1. Replace the original HASPVDD.DLL driver. 2. Set the following registry key in HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\HaspNt\Parameter: * LegacyVDDInterface = 1 3. Optionally, log traffic into a binary logfile by setting the LogFile registry value. **Dumping Logs** Logged calls can be later dumped in human-readable form using the dumplog utility. **Alternative Emulation Method** This method is not recommended anymore and serves as a reference only. It may be sufficient to return static values for 02 requests, which is detailed in the chapter about the HASPEMU.SYS driver. Alternatively, you can use the normal HASPDOS.SYS driver and let the HASPVDD.DLL driver handle emulation. This approach allows using the HASPEMU.SYS driver with other DOS emulations or even directly in good old plain DOS. To emulate the dongle using this method, you need to: 1. Sniff traffic between the application and the dongle. 2. Gather logs from it. 3. Create a registry key named EmulateParams containing the binary representation of the expected answer. 4. Use the example from HASPEMU.SYS driver as a reference. By following these steps, you can achieve emulation without relying on the HASPNT64.SYS driver. **Shortest Path to Emulation** The shortest path to emulation is using HASPEMU.SYS, followed by the second shortest approach: using this emulation interface (HASPDOS.SYS + HASPVDD.DLL).The use of user-mode dongle emulation is preferred over kernel mode drivers to minimize potential problems. By utilizing a user-mode approach, it's possible to avoid the risks associated with kernel-level interactions. In this context, the HASPVDD driver supports the use of a plugin DLL that provides emulation functions. The UCLHASP driver is a critical component of the HASP dongle support on 32-bit systems, but it poses some challenges for 64-bit systems. The underlying HARDLOCK.SYS driver on 64-bit systems expects a specific internal structure with 2 pointers as 64-bit values, which requires a different HASPVDD.DLL configuration. To use the UCLHASP driver on a 64-bit system, it's essential to understand its limitations and requirements. The driver has not been thoroughly tested and poses risks of data processing in kernel modes. Therefore, it's recommended to avoid using this interface and instead opt for the IOCTL-based interface provided by HASPVDD.DLL. The UCLHASP driver is designed primarily for DOS support, and other callers relying on different interfaces or packet sizes won't work with it. However, it's possible to compile and run this driver on a 32-bit x86 system, and it requires a working 64-bit HARDLOCK.SYS driver on the system. The package includes well-documented examples of HASP dongle workflow for applications running on DOS on Windows NTVDM. These drivers enable running HASP Dongle-protected commercial DOS applications natively on 64-bit Windows, extending the functionality of NTVDMx64. ========== Using the HASPHL_Tools.zip to dump the USB HASP, you can use h5dmp.exe to achieve this. Simply run the command "h5dmp.exe 0xPW1 0xPW2" to get the desired output. Upon successful execution, you should be able to see two files, namely hasp.dmp and hhl_mem.dmp. To create a multikey registry file from dump 1, use 'UniDumpToReg' (HaspHL_Tools.zip\UniDumpToReg\UniDumpToReg.exe). Open the created hasp dump and navigate to the same directory where you have both files. Use the 'Chingachguk based Hasp HL' tool to export a registry key. The generated reg key should be edited by changing "HKEY_LOCAL_MACHINE\System\CurrentControlSet\NEWHASP\Services\Emulator\HASP\Dump\" to "HKEY_LOCAL_MACHINE\System\CurrentControlSet\MultiKey\Dumps\". After making these changes, you'll have the necessary registry file for emulation. To emulate your HASP using multikey, plug out the USB HASP and use 'multikey64'. You can add the registry key you created earlier and install MultiKey by running the install.cmd file. Lookin for how to activate Sentinel, try this: First, go to C:\Windows\Sistem32\drivers\multikey.sys, then exit and restart your PC. Now insert your Sentinel Hardware Key and run "ssp2mk". After that, just click the icon on the top left corner. Your system should now show a connected state. Next, open regedit4 and navigate to the middle field. Save it as a .reg file, for example 33B4_86C9_0002-9003.reg. Then, just double-click this new file and confirm you want to add these changes to your System Registry. After that, run "Regedit" and make sure everything works out (HKEY_LOCAL_MACHINE / SYSTEM / CurrentControlSet / MultiKey / Dumps / key-name). Later, just remove the Sentinel hardware key and restart your computer again. Now you shouldn't need to put in that hardware key when starting up your software.