NVM Express Drives and Digital Forensics

by Bruce J. Nikkel
nikkel@digitalforensics.ch

January 29, 2016

Abstract

This paper provides an overview of NVME technology and discusses the relevance to the digital forensics community. The NVM Express standard defines an interface and command set for communication between a host and non-volatile memory devices (SSD's). It allows the direct attachment of SSD storage to the PCI Express bus using PCIe slots, M.2, and U.2 interfaces. NVME was developed with a new command set and is not compatible with ATA/ATAPI or SCSI commands. The introduction of NVME to the market creates new challenges when performing digital forensic acquisition and analysis, where legacy drive commands are expected.

Keywords:NVME, PCIE, NVMExpress, PCIExpress, SATAExpress, M.2, U.2

1 Introduction to NVM Express

The fundamental concepts of digital forensics describe the acquiring of storage media for use as evidence. This includes maximizing data completeness and minimizing data modification during the forensic acquisition process. Forensic tools and techniques for storage media interfaces such as IDE, ATA, SATA, SCSI, and SAS, are well known and tested in the forensics community. However, a new storage standard, NVM Express (NVME), is being introduced to the market which is not necessarily compatible with traditional digital forensics. The digital forensics community needs to be aware of this new storage standard, and take measures to ensure tools, techniques, and processes are adequately updated and tested.

The NVM Express standard defines an interface and command set for communication between a host and non-volatile memory devices (SSD's), attached to a system by the PCI Express bus.

The initial Enterprise NVMHCI standard was completed in 2008 by an industry workgroup, which later formed NVM Express Inc. and published the NVM Express specification in 2011. The standard was created from scratch, without consideration for legacy protocols or backward compatibility. As of this writing, the most recent version available is NVM Express 1.2a[1] and available at http://www.nvmexpress.org.

NVM Express was intended to replace the aging ATA/ATAPI[8] and AHCI[7] standards, which were originally designed for magnetic hard disks connected via cables to uniprocessor machines. The new standard assumes the use of non-volatile memory (SSD's) directly attached to the PCI Express bus, residing in systems with multiple CPU cores, and potentially using virtual machine technology. NVME provides up to 64K command queues with 64K commands for each queue, and allows multiple interrupts using MSI-X (SATA/AHCI has a single queue of 32 commands and uses one interrupt). NVME also provides new features such as support for SR-IOV (Single Root I/O Virtualization) for high performance virtual machine I/O, and Namespaces which allow low level segmenting/partitioning of a physical NVME drive. The resulting NVME standard is high performance, low latency, scalable, facilitates parallelization, and has a compact efficient command set.

It is important to distinguish between NVM Express and SATA Express. SATA Express drives also connect directly to the PCI Express bus, but continue to implement the legacy AHCI standard rather than the newer NVME standard. NVM Express and SATA Express drives may look physically similar, but are not the same. The manufacturer model specifications will indicate if the drive is implementing the NVME or AHCI standard.

2 NVME physical connectors

There are several physical connection types used to attach NVME drives to a host. These include regular PCI Express expansion slots, the M.2 PCB edge connector interface (also known as the Next Generation Form Factor or NGFF), and the U.2 cabled interface (also known as SFF-8639). A good reference for various physical SSD form factor standards is available at http://www.ssdformfactor.org which maintains the Enterprise SSD Form Factor standard[2].

Regular PCI Express slots can be used to connect NVME drives implemented as PCIE expansion cards. One of the first consumer NVME drives on the market was the Intel 750 series[3], which was developed as a PCIE card (see Figure 1). Higher capacity NVME drives are built as PCIE cards due to the larger PCB surface area available to house NVM chips. The use of PCIE slots also facilitates the use of more than four PCI Express lanes for increased throughput, compared to the four lane (x4) maximum for M.2 and U.2.

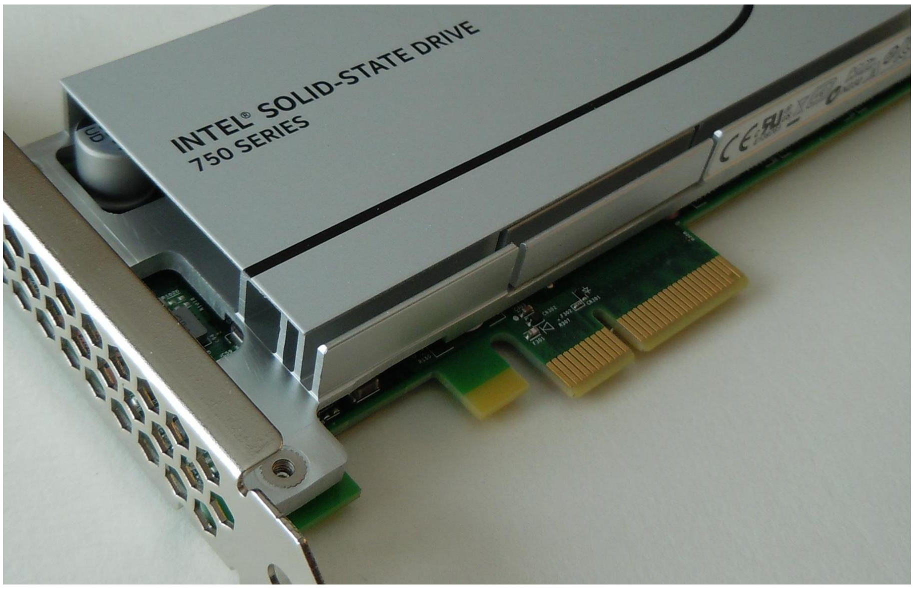

Figure 1: Intel 750 Series with PCI Express slot

The M.2 physical interface is available on newer mainboards, notebooks, and other mobile devices. This physical interface was designed for multi-functionality, and may attach traditional AHCI based SSD drives (SATA Express), NVME SSD drives, USB 3.x, wireless cards, and other peripherals. As of this writing, the majority of M.2 SSD drives on the market are still AHCI based, and not NVME. An Example of an NVME based M.2 SSD drive is the Samsung SSD 950 Pro[4], shown in Figure 2. NVME drives typically use M.2 ”type M” edge connectors, allowing them access to four PCIE lanes.

The U.2 interface for NVME SSD drives allows traditional 2.5 inch physical form factors to be connected via cable or backplane. The U.2 (SFF-8639) interface and cable is mechanically similar to SAS cabling, but with additional pins for PCIE lanes. The cable connects the drive enclosure to a mini-SAS HD plug on an M.2 adapter, which is attached to the mainboard. In figure 3, left to right, is a 2.5 inch SSD drive with a U.2 interface, U.2 to mini-SAS HD cable, and a mini-SAS HD to M.2 adapter for the mainboard.

NVME drives can also be attached to a PC using other combinations of adapters.

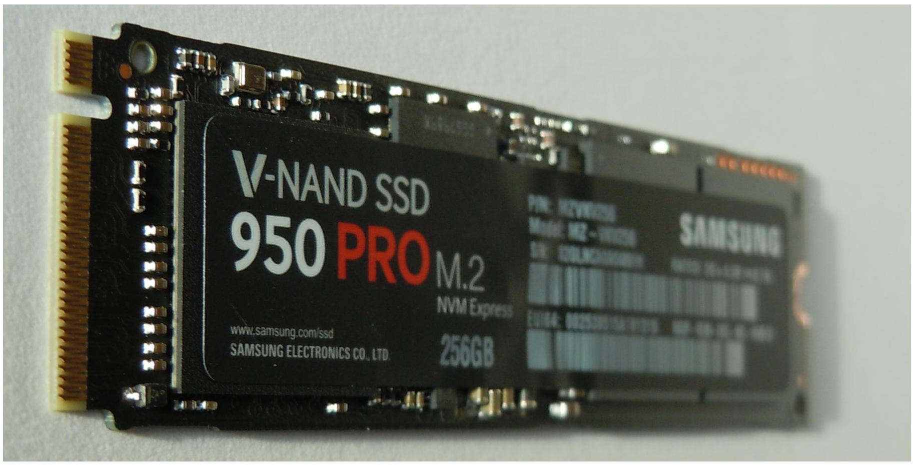

Figure 2: Samsung 950 Pro with M.2 interface

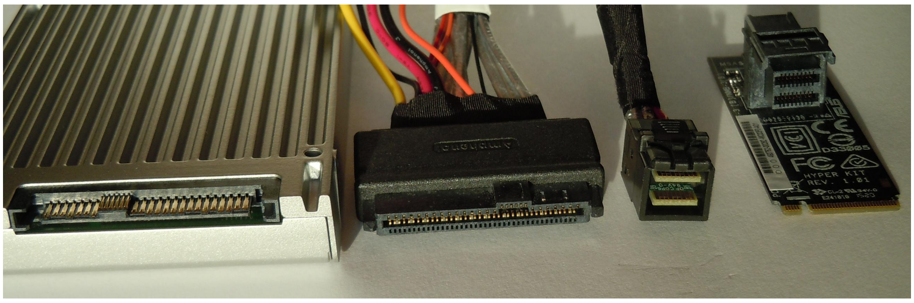

Figure 3: Intel 750 Series with U.2 interface

For example, an M.2 to PCI slot adapter is shown in figure 4.

It is important to verify the type of SSD drive by checking the model number and manufacturer specifications. Physical connectors cannot be reliably used to determine the drive type. An SSD with an M.2 interface could be a SATA-3 compatible drive, a SATA Express drive, or an NVME drive. A SAS SSD drive interface looks similar to an NVME U.2 drive interface. SSD's with a PCIE edge connector can be AHCI based, NVME, or even a non-standard proprietary drive implementation. Identifying the connector alone does not always determine if an SSD drive is based on the NVME standard.

3 NVME command sets

The NVM Express standard[1] defines an "Admin Command Set" and a "NVM Command Set" which are used to control and to communicate with the device. Commands are submitted to command queues for execution by the device. There can be up to 64K command queues, each queuing up to 64K pending commands for execution. The host submits a command through a register in-

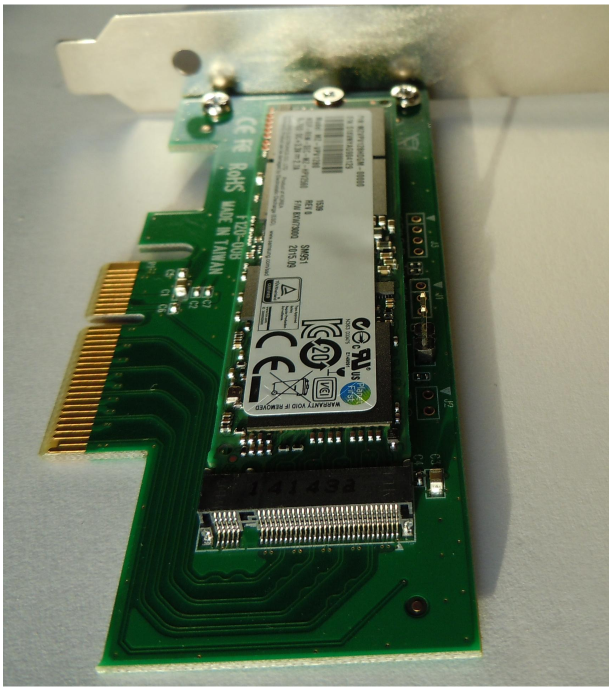

Figure 4: Samsung 951 (NMVE) and a M.2 to PCIE slot adapter

terface, and is notified through a completion queue once the command has been completed[1].

Both Admin Commands and NVM Commands are 64 bytes in size. The command format contains information about the Command Identifier (CID), various command attributes, the Namespace Identifier (NSID), pointers to data locations, vendor specific command information, and other optional command definitions.

Unlike the large SCSI and ATA/ATAPI command sets, the NVM command set was designed to be small, without the need to maintain backward compatibility or provide legacy features. The complete Admin and NVM command sets can be easily listed here.

[1] Notification of new queued commands or completed commands is done using a "doorbell" concept described in the NVME standard

The Admin Command Set (implementation of the first eight commands is mandatory):

Delete I/O Submission Queue
Create I/O Submission Queue
Get Log Page
Identify
Abort
Set Features
Get Features
Asynchronous Event Request
Namespace Management
Firmware Commit
Firmware Image Download
Namespace Attachment
I/O Command Set specific
Vendor specific
Format NVM
Security Send
Security Receive

The NVM Command Set (implementation of the first three is mandatory):

Flush
Write
Read
Write Uncorrectable
Compare
Write Zeroes
Dataset Management
Reservation Register
Reservation Report
Reservation Acquire
Reservation Release
Vendor Specific

These commands are described in precise detail in the NVM-Express standards document (1.2a was referenced for this paper). An understanding of both the Admin and NVM command sets is necessary to develop hardware and software write-blockers, and to develop NVME specific forensic software features.

A SCSI to NVME Translation Layer (SNTL) document was also created to define a mapping between NVME and some SCSI commands[5].

4 Operating system device representation

The use of NVME may require support from both hardware manufacturers and operating system developers. Vendors of mainboards need to include NVME support in their firmware if booting from an NVME device is desired. NVME device drivers are required by operating systems, as the generic SCSI or AHCI drivers can not be used with NVME hardware.

Microsoft included driver support for NVME in Windows 8.1 and Windows Server 2012 R2, and Linux support for NVME was added as of kernel version 3.3. Earlier versions may require third party drivers. Other operating systems such as FreeBSD and OSX have added support for NVME in recent operating system releases.

NVME devices are recognized with modern Linux kernels, and attach to the device tree as a single PCI function. An example of listing the attached NVME devices on a Linux system is shown here (with four attached NVME drives):

```
# lspci -d ::0108
02:00.0 Non-Volatile memory controller: Intel Corporation PCIe Data Center SSD (rev 01)
04:00.0 Non-Volatile memory controller: Samsung Electronics Co Ltd Device a802 (rev 01)
06:00.0 Non-Volatile memory controller: Intel Corporation PCIe Data Center SSD (rev 01)
0c:00.0 Non-Volatile memory controller: Samsung Electronics Co Ltd Device a802 (rev 01)
```

Here lspci lists all devices with the "01" mass storage controller device class, and the "08" non-volatile memory controller subclass[2].

NVME devices are not SATA or SCSI, and therefore not represented as /dev/sd* devices under the Linux device directory. They have an alternate file naming convention beginning with /dev/nvme*. The naming convention allows for representation of multiple devices, which may contain multiple namespaces, which in turn may contain multiple partitions. For example, a host with a single NVME drive containing one namespace with three partitions appears as follows:

```
# ls /dev/nvme*
/dev/nvme0  /dev/nvme0n1  /dev/nvme0n1p1  /dev/nvme0n1p2  /dev/nvme0n1p3
```

Here "nvme0" refers to the character device of the nvme drive, "n1" refers to the raw block device of the namespace, and "p*" refer to the three partition devices (normal partitions, created with fdisk).

Namespaces are conceptually similar to partitions, but are done at a lower layer abstracted from normal operating system activity[3].

The logical block size of NVME 'sectors' can be specified during the configuration of the device, and should be correctly detected by the kernel when the capabilities of the device are queried.

Linux was used in the examples here, other operating systems may represent NVME devices differently.

[2] Using lspci with "-d ::" ignores the vendor and device ID's.

[3] There were no NVME drives supporting multiple namespaces available for testing during the writing of this paper.

5 Tools for querying NVME devices

Drive vendors may provide proprietary management tools to configure an NVME device, create and delete namespaces, upgrade firmware, perform diagnostic tests, and other management tasks. For example, Intel provides the "Intel Solid State Drive Data Center Tool", including a Linux command line version ('isdct' shown below) to manage the Intel 750 series of devices.

```
# isdct show -intelssd
- IntelSSD CVCQ514500N2400AGN -
DeviceStatus: Healthy
Firmware: 8EV10171
FirmwareUpdateAvailable: The selected Intel SSD contains current firmware as of this tool
ModelNumber: INTEL SSDPEDMW400G4
ProductFamily: Intel SSD 750 Series
SerialNumber: CVCQ514500N2400AGN
Index: 0
DevicePath: /dev/nvme1n1
Bootloader: 8B1B0131
```

Since NVME is an open standard, generic tools can be developed which interact with NVME devices (with limited functionality for vendor specific NVME commands). An open source utility called 'nvme-cli' is available[6] for querying and managing any NVME devices. This tool is under active development and useful for listing and querying NVME drives. An example is shown here:

```
# nvme list
Node             Model                Version  Namepace Usage                      Format           FW Rev
---------------- -------------------- -------- -------- -------------------------- ---------------- --------
/dev/nvme0n1     INTEL SSDPE2MW400G4  1.0      1        400.09  GB / 400.09  GB    512   B +  0 B   8EV10171
/dev/nvme1n1     SAMSUNG MZVPV128HDGM 1.1      1          0.00   B / 128.04  GB    512   B +  0 B   BXW7300Q
/dev/nvme2n1     INTEL SSDPEDMW400G4  1.0      1        400.09  GB / 400.09  GB    512   B +  0 B   8EV10171
/dev/nvme2n1p1   INTEL SSDPEDMW400G4  1.0      1        400.09  GB / 400.09  GB    512   B +  0 B   8EV10171
/dev/nvme3n1     Samsung SSD 950 PRO  1.1      1          3.01  GB / 256.06  GB    512   B +  0 B   1B0QBXX7
/dev/nvme3n1p1   Samsung SSD 950 PRO  1.1      1          3.01  GB / 256.06  GB    512   B +  0 B   1B0QBXX7
/dev/nvme3n1p2   Samsung SSD 950 PRO  1.1      1          3.01  GB / 256.06  GB    512   B +  0 B   1B0QBXX7
/dev/nvme3n1p3   Samsung SSD 950 PRO  1.1      1          3.01  GB / 256.06  GB    512   B +  0 B   1B0QBXX7
```

Because NVME is not providing a SCSI or AHCI interface to the operating system, any tools designed to interact with drives below the operating system's virtual filesystem and block layer may fail to function as expected. Tools operating directly on the storage device need to explicitly support NVME drives. To illustrate, the smartctl tool issues ATA or SCSI commands directly to the device to fetch SMART data. Even though NVME drives can provide SMART data, this is not accessible with the tool version tested here:

```
# smartctl -a /dev/nvme0n1
smartctl 6.4 2014-10-07 r4002 [x86_64-linux-4.2.0-16-generic] (local build)
Copyright (C) 2002-14, Bruce Allen, Christian Franke, www.smartmontools.org

/dev/nvme0n1: Unable to detect device type
```

The nvme-cli tool is able to query the device correctly and fetch the SMART information:

```
# nvme smart-log /dev/nvme0n1
Smart Log for NVME device:/dev/nvme0n1 namespace-id:ffffffff
critical_warning          : 0
temperature               : 39 C
available_spare           : 100%
available_spare_threshold : 10%
percentage_used           : 0%
data_units_read           : 59
data_units_written        : 0
host_read_commands        : 3,935
host_write_commands       : 0
controller_busy_time      : 0
power_cycles              : 30
power_on_hours            : 15
unsafe_shutdowns          : 17
media_errors              : 0
num_err_log_entries       : 0
```

Forensic tools which are not directly querying the device with ATA or SCSI commands may continue to function correctly. An example from Sleuthkit's mmls operating properly on an NVME namespace is shown here:

```
# mmls /dev/nvme3n1
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors
```

```
     Slot    Start        End          Length       Description
00:  Meta    0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000000   0000002047   0000002048   Unallocated
02:  00:00   0000002048   0167774207   0167772160   Linux (0x83)
03:  00:01   0167774208   0335546367   0167772160   Linux (0x83)
04:  00:02   0335546368   0500118191   0164571824   Linux (0x83)
```

It is important that forensic software developers test their tools on NVME devices to ensure consistent operation, valid results, and to provide confirmation of compatibility for their customers.

6 Write-blocking and NVME devices

The basis of traditional forensic write blocking, hardware or software, is the interception of ATA or SCSI commands which may lead to modification of the storage media being protected as evidence. NVME introduces a new command set which is ignored by write blocking technologies focused on filtering ATA and SCSI commands. As of this writing, no NVME write-blockers were available for testing.

Hardware write blockers have historically operated as a bridge (USB-to-IDE, USB3-to-SATA3/SAS, Firewire-to-SCSI, etc.) and were able to intercept or filter potentially dangerous (from a forensics perspective) commands from being sent to a drive. NVME attaches directly to the PCI Express bus, making it more difficult to use a separate adapter or cabling to block commands at lower level protocol layers.

Directly intercepting commands between the NVME device and the PCIE bus could require the capture and decoding of lower layer PCIE protocols (TLP packets, DLLP packets, etc). Possibilities for inserting write-blocker functionality could include the use of Thunderbolt (which acts as a PCIe bridge), the M.2 interface, or PCI Express Cards. This is an area which needs further research and development.

Write-blocking functionality might be easier to implement in software. An imaging host booting from non-NVME drives such as CDROM, SATA, or SAS, could implement a patched NVME driver which performs global write blocking on all attached NVME devices.

7 Forensic acquisition and namespaces

Forensic imaging of NVME devices can be performed in a similar manner as other sector based storage. If the correct device is chosen, current acquisition software should be able to acquire the individual blocks. For example, here an NVME drive is acquired using the familiar dcfldd tool:

```
# dcfldd if=/dev/nvme1n1 of=nvme-image.dd
3907328 blocks (122104Mb) written.
```

```
3907338+1 records in
3907338+1 records out
```

The concept of NVME namespaces is important when developing forensic tools and performing forensic acquisition. When multiple namespaces exist, each one needs to be imaged and analyzed separately.

Standard forensic acquisition processes include checking for the existence of an HPA and DCO. These do not exist on NVME drives and can be ignored, but a check for the existence of multiple namespaces should be performed. Namespaces are not like DCO and HPA, and should not be ”removed”. Attempting to remove a namespace may irrevocably destroy evidence on the NVME drive.

Consumer NVME devices on the market today typically only support a single namespace. During the writing of this papers, no devices were tested that support the creation of multiple namespaces[4].

The number of namespaces supported and used can be checked by sending an identify controller admin command. In the following example, various information about namespace support is shown:

```
# nvme id-ctrl /dev/nvme1 -H
NVME Identify Controller:
vid     : 0x144d
ssvid   : 0x144d
sn      : S2GLNCAGA04891H
mn      : Samsung SSD 950 PRO 256GB
fr      : 1B0QBXX7
...
oacs    : 0x7
  [3:3] : 0 NS Management and Attachment Not Supported
...
  [0:0] : 0x1 SMART/Health Log Page per NS Supported
...
nn      : 1
...
```

The ”Optional Admin Command Support” (OACS) indicates that namespace management is not supported on this particular drive. The ”Number of Namespaces” (NN) field shows the number of namespaces on the controller, one single namespace for this particular device.

The size of the namespace can also be checked using nvme-cli and compared with the manufacturer’s specifications:

```
# nvme id-ns  /dev/nvme0n1
NVME Identify Namespace 1:
nsze    : 0x2e9390b0
ncap    : 0x2e9390b0
nuse    : 0x2e9390b0
...
```

[4] Improved namespace management was introduced in NVME Express 1.2, and the consumer drives tested here were based on 1.0 and 1.1

Here NSZE refers to the Namespace Size, NCAP is the Namespace Capacity, and NUSE is the Namespace Utilization.

A third check can be simply listing the devices for the existence of multiple namespace devices (/dev/nvme0n2*, /dev/nvme0n3*, etc) as detected by the operating system.

As devices supporting multiple namespaces become more readily available on the market, further research will be useful to understand the precise relevance to digital forensics. As of this writing, hardware supporting the management of namespaces was difficult to obtain, and comments here are based on reviewing the NVM-Express 1.2a standards document (not actual testing). There could be potential for attempted data hiding using multiple namespaces. This may affect how forensic imaging is conducted, likely requiring separate images to be made for each namespace.

In virtual hosting environments where separate namespaces are allocated for each virtual machine (using SR-IOV), it may be possible to selectively image only the namespace of the suspected VM, rather than the entire drive (ignoring other VMs not in scope of an investigation).

Forensic tool developers need to include checking for multiple namespaces when adding NVME support to their products. The current expectation in the forensics community is that one physical drive yields one image file. The possibility of multiple namespaces on a single drive changes this assumption.

8 NVME forensic artifacts

NVME drives may have additional artifacts of interest to forensic examiners. Some forensic artifacts mentioned here are taken from the NVM-Express 1.2a standard. Some of these may not be supported by all NVME drives, and different tools (possibly proprietary) may be needed to perform the queries.

Lists of various configuration details and other information about the controller can be produced using the nvme-cli tool.

```
# nvme id-ctrl -H /dev/nvme0
NVME Identify Controller:
vid     : 0x8086
ssvid   : 0x8086
sn      : CVCQ526600BS400CGN
mn      : INTEL SSDPE2MW400G4
fr      : 8EV10171
rab     : 0
ieee    : 5cd2e4
cmic    : 0
  [2:2] : 0 PCI
  [1:1] : 0 Single Controller
  [0:0] : 0 Single Port
...
```

This output describes the make, model, serial number, and firmware revision of the device. It contains other information such as the namespace configuration, feature sets, queue sizes, security features, and various vendor specific details.

Controller information is "global" to the device, and additional information can be queried for individual namespaces. To list information about a specific namespace in human readable form:

```
# nvme id-ns -H /dev/nvme0n1
NVME Identify Namespace 1:
nsze    : 0x2e9390b0
ncap    : 0x2e9390b0
nuse    : 0x2e9390b0
nsfeat  : 0
  [2:2] : 0 Deallocated or Unwritten Logical Block error Not Supported
  [1:1] : 0 Namespace uses AWUN, AWUPF, and ACWU
  [0:0] : 0 Thin Provisioning Not Supported

nlbaf   : 6
flbas   : 0
  [4:4] : 0 Metadata Transferred in Separate Contiguous Buffer
  [3:0] : 0 Current LBA Format Selected

mc      : 0x1
  [1:1] : 0 Metadata Pointer Not Supported
  [0:0] : 0x1 Metadata as Part of Extended Data LBA Supported
```

This output provides configuration and attributes for each namespace. It describes thin provisioning, protection information, format progress, block size in use, and other information specific to the specified namespace.

It is also possible to send individual commands to retrieve features from a device. These commands may be derived from the NVM Express standard, or contain proprietary commands specific to a particular NVME product. The example here demonstrates the requesting of a particular feature:

```
# nvme get-feature /dev/nvme0 -f 7
get-feature: 0x07 (Number of Queues), Current value: 0x1e001e
```

Various SMART details can also be requested. Standard SMART log output was shown in a previous example, additional SMART output is shown here:

```
# nvme smart-log-add  /dev/nvme0
Additional Smart Log for NVME device:/dev/nvme0 namespace-id:ffffffff
key                               normalized raw
program_fail_count              : 100%       0
erase_fail_count                : 100%       0
wear_leveling                   : 100%       min: 1, max: 4, avg: 2
end_to_end_error_detection_count: 100%       0
crc_error_count                 : 100%       13
timed_workload_media_wear       : 100%       0.000%
timed_workload_host_reads       : 100%       100%
timed_workload_timer            : 100%       1214 min
```

```
thermal_throttle_status         : 100%       0%, cnt: 0
retry_buffer_overflow_count     : 100%       0
pll_lock_loss_count             : 100%       0
nand_bytes_written              : 100%       sectors: 3463
host_bytes_written              : 100%       sectors: 0
```

Checking with the vendor documentation may provide further proprietary commands to query for information of interest in a forensic context.

It is also possible for individual namespaces to have separate attributes, independent of the other namespaces on an NVME drive.

9 Deallocated blocks, wiping, and security

NVM Express drives are SSD's, and exhibit the same lower level attributes as SATA/AHCI SSD drives. They have a Flash Translation Layer (FTL), perform over-provisioning, wear leveling, etc. This is abstracted from the NVME standard. Many of the same digital forensic concerns with SSDs[13] apply to NVME drives.

NVME drives have a 'deallocate' command which functions similar to the ATA TRIM command. Operating systems which issue deallocate commands will destroy data in unallocated areas of the drive.

SSD drives can be quickly and easily wiped. A suspect with enough advance warning can issue several types of commands to wipe a drive, destroying evidence. The NVME standard includes the "Format NVM" command which can be used to erase a single namespace or all namespaces.

Formating a drive with nvme-cli quickly erases the drive, destroying evidence[5]. An example is shown here:

```
# nvme format /dev/nvme1n1
Success formatting namespace:1
```

Another method of destroying the data on encrypted drives is simply to wipe the encryption key, rendering the data inaccessible. The standard refers to this as "Secure Erase" or "Cryptographic Erase".

The NVME standard does not specify a new set of security features, but relies on existing standards defined by the Technical Committee T10[11]. The implementation of security features is dependent on the manufacturer of the NVME device.

10 Conclusion and digital forensic challenges

This paper raises awareness of the arrival of NVME drives on the consumer market, and highlights new challenges for the digital forensics community.

[5] The specification requires a controller not to return previously written data after a format, but it does not guarantee secure destruction of data below the FTL

Forensic labs need to be aware of NVME technology and update processes to include the processing of NVME drives, checking for multiple namespaces, and gathering available artifacts which are unique to NVME drives.

Software vendors need to test their products to ensure compatibility and proper behavior with NVME devices. The assumption cannot be made that all current forensic software will automatically work as expected with NVME devices. Developers may need to add additional features to probe and access new forensic artifacts associated with NVME technology. Vendors need to be able to provide confirmation and assurance to their customers that NVME technology is supported in their products, and producing consistent and valid results.

Hardware write-blocker manufacturers have a challenge to develop new products that perform write-blocking of NVME devices, intercepting commands passing over the NVME interface on the PCI Express bus.

Software write-blocker developers also have a challenge to update their software to support the NVME command set (keeping in mind, there is not just a single command queue like SATA, but up to 64K command queues).

Forensic standards bodies such as NIST CFTT[12] need to create similar testing procedures for NVME write-blocking tools, and be ready to validate NVME write-blocking forensic products as they come to market.

As of this writing, it is unclear how popular NVME drives will become in the future, especially in the the consumer market place. Currently most PCI based storage devices are still implemented using the AHCI standard which supports ATA commands. However, the performance and efficiency benefits of NVME are compelling, and will become more advantageous as SSD capacities increase and the cost of production drops.

References

[1] NVM Express, Revision 1.2.a, nvmexpress.org, October 23, 2015

[2] Enterprise SSD Form Factor, Version 1.0a, ssdformfactor.org, December 12, 2012

[3] Intel Solid State Drive 750 Series Product Specification, Revision 004, October 2015

[4] http://www.samsung.com/950PRO, fetched 2015-11-14.

[5] NVM Express: SCSI Translation Reference, Revision 1.5, nvmeexpress.org, June 24, 2015

[6] nvme-cli, NVM-Express user space tooling for Linux, https://github.com/linux-nvme/nvme-cli, fetched 2015-11-15

[7] http://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/serial-ata-ahci-spec-rev1-3-1.pdf, fetched 2015-11-15

[8] ATA/ATAPI standards, http://www.t13.org

[9] PCI Express standards, https://pcisig.com/specifications/pciexpress/

[10] SCSI standards, http://t10.org/drafts.htm

[11] ISO/IEC 14776-454, SCSI Primary Commands - 4 (SPC-4), Available from http://www.t10.org

[12] NIST Computer Forensics Tool Testing (CFTT) Program, http://www.cftt.nist.gov

[13] Graeme B. Bell, Richard Boddington - Solid State Drives: The Beginning of the End for Current Practice in Digital Forensic Recovery? Journal of Digital Forensics, Security and Law 5,2011

Acknowledgement: Thanks to Keith Busch from Intel for his feedback on the draft of this paper.