Lessons Learned
“Implementing Support For External Storage of Sealed Data”

Zach Champoux
04/11/2017

Let this document serve as a “lessons learned” cookbook for implementing support for external storage of sealed data, using a modified Grub2 source and the Syracuse Assured Boot Loader Executive (SABLE) on a platform running Ubuntu version 16.04 LTS. Development of this feature aims to address the scenario where a system requires the storage of more SABLE-Enabled Configurations (SEC) than its TPM's NVRAM capacity can support. This feature provides a means of storing the SECs on a different storage medium, specifically a Hard Disk Drive (HDD). Many, or all, of the steps provided in this cookbook may apply, as they are written here, in other environments as well, but Ubuntu version 16.04 is the environment in which I've tested the following steps and proven their validity.

NOTE: The feature discussed in this document is currently at an early stage of development. The implementation described serves as a concept design, purposed for demonstrating the possibility of implementing such a feature. My thoughts for future development of this feature are provided at the end of this document.

Environment specifications:

Ubuntu version 16.04 LTS (Long Term Support)
kernel version 4.4.0-71-generic
GRUB version 2.02~beta3 (acquired from the GNU Git repository)

TPM NVRAM space permissions:
“AUTHWRITE” and “READ_STCLEAR”

Modified Grub 2:

The source code for my modified Grub 2 is on the Git server, at the following location:

'Jacques/DARPA/modified_Grub2_external_storage_170411_CLEAN'

The modifications are located in the following files:

'[Top Source Directory]/grub-core/hello/hello.c'
'[Top Source Directory]/grub-core/loader/multiboot_mbi2.c'

The 'hello' command has been modified to serve several functions. When the 'hello' command is run from the Grub 2 command line:

1.) The sealed data is read from a specified NVRAM space within the TPM.

2.) The sealed data is written to the HDD, utilizing the BIOS 13h interrupt
3.) The sealed data is read from the HDD, utilizing the BIOS 13h interrupt
4.) Values are placed in global variables to be used in 'multiboot_mbi2.c', where the re-purposed module is added

The MBI Structure:

… in 'multiboot_mbi2.c'

5.) Global variables, whose values were assigned in 'hello.c', are used to create a re-purposed module to be received by SABLE when it loads, allowing SABLE to access the sealed data.

The 'hello' command must be run, each time, before SABLE is loaded via the Grub 2 command line. To return to the Grub 2 menu from the command line after running 'hello', run the command, 'normal'.

Modified SABLE:

The source code for the modified SABLE is on the Git server, at the following location:

'Jacques/DARPA/sable_external_storage_170411_CLEAN'

A few minor code extensions were made to 'sable.c' to acquire the sealed data, to be used in a 'measure', from the re-purposed module, rather than via 'TPM_NV_ReadValue()'.

As it is loading in the modules from the Multiboot Information (MBI) structure, SABLE detects the re-purposed module (sent by Grub 2) by looking at the value of the 'string' field in each module and comparing it to a predefined flag (The flag name should be unique in value). In this case, the flag is 'read'.

Storing The Sealed Data On The HDD:

In choosing a location on the HDD to store the sealed data 'blob', I must identify an area on the HDD that is unused. I do not want my sealed data being overwritten by some other software. Disk I/O is performed utilizing a Grub 2 facility that executes the BIOS 13h interrupt. Though the 'fdisk' utility lists my HDD as having 4,096-byte physical sectors, my BIOS writes to the disk in 512-byte chunks.

Storing the sealed data requires actual low-level disk I/O, something that is not possible on areas of the HDD where Logical Block Addressing (LBA) is employed (e.g. a Linux partition). I look for an area on the HDD located past my Grub 2 image, but before the first disk partition. In doing so, I utilize the 'fdisk' utility, which tells me the sector at which my first disk partition starts. Since my first partition starts at the 'fdisk'-provided default, sector 2,048, I know the sector I use to store my sealed data must be located before sector 2,048. As for my Grub 2 image, I must determine where the image is being written on the disk, and how many sectors it occupies. To do this, I run the following 'hdparm' command to write zeros to sectors 1 through 2,047:

```
for i in {1..2047}; do sudo hdparm –yes-i-know-what-i-am-doing –write-sector $i /dev/sda; done
```

I then run a Grub 2 install using the following command:

```
sudo /usr/sbin/grub-install /dev/sda
```

I then run the following command to read sectors 1 through 2,047 to determine where, on the disk, the Grub 2 image has been written.

```
for i in {1..2047}; do sudo hdparm –read-sector $i /dev/sda; done
```

The results reveal the Grub 2 image to have been written to sectors 1 through 102.

After repeating the above described process several times, I determine that my Grub 2 image is consistently being written to sectors 1 through 102 on every Grub 2 re-install. On every Grub 2 re-install, sectors 103 through 2,047 remain 'untouched'. At this point, I believe it is a reasonable assertion that, given my current system and configuration, sectors 103 through 2,047 are currently not being used by any software. As far as choosing a sector to write my sealed data 'blob' to, I wanted to aim somewhere in the middle between sector 103 and sector 2,047, so I decided on sector 1,000. Sector 1,000 is a considerable distance away from sector 102, allowing some room for my Grub 2 image to grow and still not overwrite the sector containing my sealed data 'blob'. Given my research and observations up to this point, I consider this sector to be a reasonably safe area for storage.

NOTE: Though I have determined this particular sector to be a "safe", previously unused area on my HDD, this may not be the case on other systems; especially those systems with a bigger Grub 2 image.

The BIOS disk I/O subroutine (i.e. interrupt 13h) may be utilized when the CPU is operating in 16-bit Real Mode. Grub 2, which provides a facility that utilizes the BIOS disk I/O subroutine, runs in Real Mode. My process for writing the sealed data to the HDD sector, within Grub 2, is as such:

I begin with a char array storing my sealed data…

```
char sealed_Data[512];
```

The BIOS 13h (disk I/O) interrupt requires that I specify the 16-bit segment and 16-bit offset of the physical address in RAM where my data to be written (i.e. 'sealed_Data' char array) is located. Since the processor is running in Real Mode, the value of 'sealed_Data' is the actual 20-bit physical address in RAM where the first element of the array is located (virtual addressing is not enabled until Protected Mode). I determine the segment and offset by doing the following:

```
unsigned int segment = 0;
unsigned int offset = 0;
unsigned int bitwise_Zero = 0;

segment = (unsigned int)sealed_Data >> 4;
offset =  (unsigned int)sealed_Data & (~(~bitwise_Zero << 4));
```

The Grub 2 function that utilizes the BIOS disk I/O facility is called 'grub_biosdisk_rw_standard()'.

```
grub_biosdisk_rw_standard (0x03, 0x80, 0, 15, 56, 1, (int)segment, (int)offset)
```

The first argument specifies the hex value of the function to be performed; 0x03 indicates the function for writing sectors to disk. The second argument specifies the HDD. The third, fourth and fifth arguments specify the cylinder number, sector number and head number, respectively. The sixth argument specifies the number of 512-byte sectors to be written. The seventh and eighth arguments specify the segment and offset, respectively, of the data to be written.

As mentioned previously, I determined sector 1,000 to be a safe location on my HDD to store my sealed data. To determine the Cylinder-Head-Sector (CHS) address to access that particular sector, I first run the 'hdparm' utility to learn the geometry of my HDD.

```
#sudo hdparm /dev/sda

/dev/sda:
…
geometry        = 60801/255/63, sectors = 976773168, start = 0
```

Using this information, I can determine the CHS address of sector 1,000.

> 63 sectors/head
>
> 63 * **15** = 945
> (I want sector number 1,000 and the sector count starts at 0)
> 1001 – 945 = **56**
>
> I have determined a CHS address of 0-15-56.

'hdparm' is a useful utility for reading and writing disk sectors from the command line. 'hdparm' may prove useful in implementing a mechanism to backup the sealed data blobs stored on the HDD.

Thoughts For Future Development:

As stated at the beginning of this document, the implementation described above serves as a concept design, purposed for demonstrating the possibility of supporting external storage of the sealed data 'blobs'.
In the implementation described above, the sealed data is read from the TPM NVRAM space, written to the HDD, and read from the HDD, each time the 'hello' command is run. This redundant sequence is done only to show that each individual step can be performed. As this feature is developed to meet the practical needs of customers, these individual steps will be used only as needed. For instance, given a system that uses and stores only a single SEC at any given time, if the sealed data for that SEC has already been written to the HDD, there is no need to read from the TPM NVRAM space or write it to the HDD again. To detect when a new SEC has been created, I propose that Grub 2 use a second NVRAM space on the TPM that serves as a 'scratch pad area' where SABLE can signal to Grub 2 (via some type of a flag) when a new SEC has been created. Of course, the situation in which this particular feature would be truly valuable is one in which the system houses more SECs than its TPMs NVRAM capacity can support. For this type of system, the flag used within the second TPM NVRAM space could be the 'index' (i.e. 'index' in the context of NVRAM space) corresponding to the newly created SEC. The use of the index would provide a reasonable means for organizing the sealed data 'blobs' on

the HDD.

Useful Resources:

https://en.wikipedia.org/wiki/INT_13H