EFI TBOOT

Starting Point

- Project to date is a proof of concept.
- Initial approach (Plan A) thought up by myself and James McKenzie (Br).
- TBOOT itself is an EFI boot loader.
- TBOOT then must interact with Xen as an EFI module.
- Ideal approach would be to merge TBOOT into Xen and have a single EFI module.
- Licensing and political issues would prevent the ideal approach.

Basic Environment

- TBOOT built as a 64b PE EFI module.
- Compiled and linked using MinGW (though it can be compiled with GCC directly).
- Uses gnu-efi headers but not the library.
- Completely relocatable PIC code.
- Majority of code taken directly from current TBOOT project (v1.9.5)*.
- Majority of code is unaltered, some bits had to be altered for 64b compilation (e.g. asm code)*.

Plan A

- TBOOT loads as the boot loader. It relocates itself to a chunk of EFI Runtime Code memory and registers an EFI runtime variable.
- It does most of the preliminary setup that original TBOOT does (loading ACMs, validating the platform, validating configurations, etc).
- TBOOT loads Xen as an EFI module and starts it using boot services.
- Xen starts in its standard EFI entry point and does most of what normal EFI Xen does.

Plan A Cont.

- Xen locates the TBOOT EFI variable and acquires its config file from TBOOT.
- After Xen executes EBS it sets up a structure to pass to TBOOT containing information needed by TBOOT.
- Xen calls back into TBOOT.
- TBOOT then executes SENTER and enters the MLE.
- TBOOT proceeds with the normal post measured launch and when done, vectors back to Xen to finish booting normally.

Plan A Pros and Cons

- P: It is the least invasive in terms of modifying Xen.
- C: It is rather complicated.
- C: The Xen module cannot be measured in its entirety because it has changed since initially loaded.
- C: Xen does a lot of setup pre-measured launch that is not easily measured or validated (some thoughts on this).*

Plan B

- TBOOT loads as the boot loader. It relocates itself to a chunk of EFI Runtime Code memory.
- It does most of the preliminary setup that original TBOOT does (loading ACMs, validating the platform, validating configurations, etc).
- TBOOT reads Xen's configuration and loads all the rest of the modules.
- TBOOT gathers all pre-EBS information that Xen needs.
- TBOOT does EBS and then SENTER entering the MLE.

Plan B Cont.

- TBOOT proceeds with the normal post launch.
- Xen is modified to have a special entry point to inter-operate with TBOOT. TBOOT locates this entry point.
- TBOOT sets up the hand-off information that Xen needs and vectors to Xen to complete booting normally.

Plan B Pros and Cons

- P: Much simpler design.
- P: Much more in line with how a TBOOT measured launch works today.
- C: TBOOT must duplicate a lot of what Xen does pre-EBS.
- C: Much less likely to get Xen changes upstreamed with the odd exported function.

Issues Identified

- There is information that needs to be gotten pre-EBS that is used post-launch which is not measured or verified.
- This can be minimized but some things like video information have to be gotten pre-EBS.
- RT services cannot be trusted post-launch*.
- A bigger deal is whether other important information that Xen and dom0 need like ACPI tables/code, SMBIOS, etc are being measured.*
- It is not clear the memory validation information in the TXT heap correctly covers the EFI memory map.

Current State

- In both Plan A and Plan B, the code doing all the pre-launch work and doing the measured launch is working.
- The post-launch code is not complete.
- None of the S3 code is in place.
- Part of the Xen code for Plan A (in patches) is in place and working.
- Only the Xen code for Plan B to get the linker to export the special TBOOT entry point is in place.
- The code is here*: https://github.com/rossphilipson/efi-tboot