Contents

1 Configuration

1.1 Installing OpenXT

Prerequisites:

- In addition to standard OpenXT requirements, the machine must have HDD with GPT.

Unzip content of provided openxt-efi-20150430.zip archive to a FAT32 formatted USB drive and boot from it.

1.2 Creating Windows VM

This section describes creation of Windows 10 VM and configuring Credential Guard on it.

1.2.1 Option 1 (slow)

- create a VM called win10efi with 80Gb disk
- add another 1Gb disk
- type at the console

```
xec-vm -n win10efi -x cpuid '[ "0x1:ecx=0xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
"2147483655:edx=xxxxxxxxxxxxxxxxxxxxxxx1xxxxxxxx" ]'
xec-vm -n win10efi -x nestedhvm true
xec-vm -n win10efi -x viridian false
xec-vm -n win10efi -x extra-xenvm "nomigrate=1"
xec-vm -n win10efi -x ovmf true
xec-vm -n win10efi -k 2 -x virt-path sdg
```

- add a cdrom with windows
- start the VM tapping space, when EDK config show up choose
  - device manager
  - ovmf platform configuration
  - change preferred resolution
  - pick your monitors resolution
  - then F10 ESC ESC and continue, don't press a key to boot windows
  - and let it get to the EDK II shell
- type “reset” at the shell and press a key to boot windows when prompted
- wait about 10-30 mins for boot.wim to load

1.2.2 Option 2 (fast)

- create a VM called win10efi with 80Gb disk
- add another 5Gb disk
- add another 1Gb disk
- type at the console

```
xec-vm -n win10efi -x cpuid '[ "0x1:ecx=0xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
"2147483655:edx=xxxxxxxxxxxxxxxxxxxxxxx1xxxxxxxx" ]'
xec-vm -n win10efi -x nestedhvm true
xec-vm -n win10efi -x viridian false
xec-vm -n win10efi -x extra-xenvm "nomigrate=1"
xec-vm -n win10efi -x ovmf true
xec-vm -n win10efi -k 3 -x virt-path sdg
```

- start the VM tapping space, when EDK config show up choose
  - device manager
  - ovmf platform configuration
  - change preferred resolution

    - pick your monitors resolution
    - then F10 ESC ESC and continue, don't press a key to boot windows
    - and let it get to the EDK II shell

- then at the console find guid of 5Gb disk

```
root@xenclient-dom0:/# xec-vm -n win10efi -k 2 -g phys-path
/storage/disks/47ee69e0-6397-4e40-b0df-a4bd4b127f53.vhd
```

- find the device number of the guid

```
root@xenclient-dom0:/# tap-ctl list
...
   19286  8    0        vhd /storage/disks/47ee69e0-6397-4e40-b0df-
a4bd4b127f53.vhd
...
root@xenclient-dom0:/# cat fat32_windows_image > /dev/xen/blktap-2/tapdev8
```

- then type reset at the EDK II shell, and tap space, from EDK config menu   pick boot manager and the "EFI Hard Disk 1"
- windows will boot in 10-30 seconds

1.3 Credential guard configuration

- start optionalfeatures.exe
    - enable Hyper-V
    - for Windows 10 version 10.0.10586 (TH2) enable “Isolated user mode”
- start gpedit.msc and open “Administrative templates” -> “System” -> “Device Guard” -> “Turn On Virtualization Based Security”
    - Set "Enabled"
    - Set “Secure Platform Security Level” to “Secure Boot”
    - Set “Credential Guard Configuration” to “Enabled Without Lock”
    - Apply the changes by clicking “OK”
- reboot

2 Patch list

All attached patches are against the state of the repositories snapshotted on 2017 Apr 25

```
git://github.com/OpenXT/openxt.git
       c29598f4e54704e4152f41c12adeb73df96faff6

git://git.openembedded.org/bitbake.git
       bc57798ff39cae5ffea194c867e07136f7b6f3ec

git://git.yoctoproject.org/meta-intel
       9385d9771567a7f64fe068ca773de8661a2c90ad

git://git.yoctoproject.org/meta-java
       a73939323984fca1e919d3408d3301ccdbceac9c

git://git.openembedded.org/meta-openembedded
       2ea8d7f54a061e902657c4f8ea1f7f7c25c6c4e1

https://github.com/OpenXT/meta-openxt-haskell-platform.git
       b64dddb1fdbfdd6817144a6ce180495bc9e16102

https://github.com/OpenXT/meta-openxt-ocaml-platform.git
       537299fe59823a84b592fd79543dd06afff68e60

git://git.yoctoproject.org/meta-selinux
       4c75d9cbcf1d75043c7c5ab315aa383d9b227510

git://git.yoctoproject.org/meta-virtualization.git
       042425c1d98bdd7e44a62789bd03b375045266f5

git://git.openembedded.org/openembedded-core
       a9db40da62c13b0010ce5afc1fde16d987bdfbc6

git://github.com/OpenXT/xenclient-oe.git
       628a6d2de1968be187e166242f95da22d76cb78d
```

The patches are in 4 groups, each patch file contains inline comments and descriptions at the top.

The first group fix bugs we encountered in the build:

The whole point of OE is to avoid host->target contamination, and the need to maintain a large array of different host systems. It appears that some people are building XT in controlled containers so that they may safely pollute the target with host binaries.

We built without containers on a 64 bit Centos 7 host.

We compiled a 64 bit binary of binutils and gcc that targeted _only_ 32 bit systems, and we used that to build a 64 bit binary of ghc which targeted _only_ 32 bit systems.

With those in the path and using a linux32 personality it then possible to build XT without a container.

The other advantage of this approach is that recipes will fail whenever they are contaminated by host binaries, as the architecture is incompatible.

We also considered it important that if the endpoint should have SELinux then the build host should also. The current build system in unable to operate under SELinux, so we fixed that.

- build/0001-no-64-bit-runs
- build/0002-site-config
- build/0003-explicit-external-cc-for-xen

- build/0004-uim-missing-include-from-host
- build/0005-32-bit-native-icedtea
- build/0006-psuedo-on-selinux
- build/0007-more-threads

The next patch fixes bit rot in the nesting support that has occurred since we last looked at OpenXT. Most of the problems are collateral damage from the toolstack change, but there's a nasty vEPT implementation bug in Xen. We provide a fix (purely to demonstrate where the problem is) but we would not recommend using it for production as it does not enforce all the isolation requirements. You can either use our correct fix from the previous code drop, or, better, we will work with Citrix to get the problem fixed upstream as we have done with the rest of the nesting issues thus providing a sustainable solution.

- nesting/0100-repair-bitrot-in-nesting-support

The next set of patches implement the work required: adding EFI support to guests, NVRAM support for EFI variables to guests, GPT support for the installer and EFI boot support for Xen. As the implementation for tboot is still in a state of flux, we have provided an adaptor to enable booting the existing multiboot modules in an EFI environment. As currently in BIOS installations, grub's menus are disabled, we have not provided a way of editing the boot configuration at runtime. Secure Boot signing, would anyway, proscribe any such facility. The original scope of the project did not cover support for tboot, but we felt this was an elegant solution so included it.

- efi/0200-ovmf
- efi/0300-installer-gpt-support
- efi/0400-efi-boot

This group of patches also contain, but do not apply, experimental support for a new hostbridge for Xen that is a much better match for modern systems. It's feature complete, and works well, but a bug in QEMU's AHCI implementation limits its current usefulness with windows guests. Q35 northbridges have 6 AHCI interface for attaching disks. This functionality is disabled by default.

Lastly in research looking at alternative ways of booting under EFI we have some patches (not to be applied for the build) to use the canonical 64bit dom0 with EFI and Xen EFI support. We're not convinced that this approach is a good one as it exposes Dom0 to EFI.

- research/1000-research-enable-efi-support-in-xen
- research/1010-research-64on32-support-to-make-64bit-dom0

3 Verification

Verification has been similar to what has been done for previous deliverables.

3.1 Test cases.

Test suit used here has been developed for stability and performance testing of uXen VMs. In this particular case, it's been adopted for Hyper-V VMs for generating heavy CPU and memory workloads to provoke potential stability issues (like FPU state corruption).

In addition to the above, we also tested if Credential Guard gets correctly installed and running.

Note: test suit is not provided as deliverable.

All L1/L2 configurations were set up with 4 vCPUs.

All L3 configurations were set up with 2 vCPUs.

- L1/L2 HyperV and Windows 10 Enterprise x64 TH2 (10.0.10586)
    - L2/root domain: run set of tests to exercise L2 stability
    - L3, Debian Wheezy (3.2.0-4-amd64): standard installation, manual verification of system stability
    - L3, Win10e x64 TH2: standard installation, execute test suite to verify L3 stability
- L1/L2 HyperV and Windows 10 Enterprise x64 RS1 (10.0.14393)
    - L2/root domain: run set of tests to exercise L2 stability
    - L3, Debian Wheezy (3.2.0-4-amd64): standard installation, manual verification of system stability
    - L3, Win10e x64 RS1: standard installation, execute test suite to verify L3 stability

3.2 Hardware used.

All the systems have minimum 8 GB of RAM.

Following hardware have been used for testing:

- Intel Core i5-6500 (HP 280 G2 MT, TPM v1.2)
- Intel Core i5-6500 (Dell OptiPlex 3040, TPM v2)

4 Secure Boot and non-OEM/OSV UEFI applications

Secure Boot prevents the launching of unknown boot loaders, firmware applications or OSes. When enabled it verifies signatures of each boot component being loaded.

Secure Boot uses databases of known-good and known-bad signatures, keys and hashes controlled by the firmware and kept in Non Volatile memory (in UEFI secure variables). For UEFI software to be loaded it has to be either signed by a known-good key or its signature or hash has to be on known-good list.

If verification of all components is successful system boots and firmware passes control to the operating system.

Secure Boot is a hard requirement for Microsoft Credential Guard and OpenXT has been extended to support it, both booting the OpenXT platform using Secure Boot, and allowing guests to boot using Secure Boot.

PC manufacturers include Microsoft keys and signatures in their firmware so as to enable easy deployment of Windows.

The known-good and known-bad databases are respectively called db (signatures database) and dbx (forbidden signatures database). db and dbx can be updated with an authentication descriptor signed with one of KEK keys (Key Exchange Key Signature Database). And so KEK can be updated by an authentication descriptor signed with PK (Platform Key). PK is usually provisioned with an OEM-owned key. The KEK database usually contains OEM and Microsoft keys. db and dbx are usually provisioned by OEM and Microsoft signatures.

With manual interaction with the PC console it is possible to manage Secure Boot keys. This is essential for development/test purposes. A security conscious organization might wish to generate their own signing key and deploy it to their machines and even delete the Microsoft and/or OEM keys to lock the machine down. Having provisioned such a key, the machine would be able to directly boot a payload signed by the organization. Entering UEFI Setup Mode clears the PK and all the secure variables can be modified without authentication checks. For convenience, we have provided source code for an enroll tool into which the relevant keys can be embedded and automatically provisioned when the module executes, for example by placing it on a USB stick and selecting to boot from it.

In practice, at least for already deployed machines, most organizations would find the manual intervention to provision new keys too burdensome. Fortunately, an alternative mechanism is usually possible. Most machines come provisioned with an additional Microsoft-controlled key called the Microsoft Corporation UEFI CA. Microsoft run a program that enables 3rd party vendors that meet various legal and technical requirements to create UEFI programs and submit them to Microsoft for signing which they will consider after completing a review.

4.1 Shim

A common use for the Microsoft Corporation UEFI CA is for enabling 3rd party OSes to boot. A simple EFI application known as “shim” [1] is typically used for this purpose. Use of “shim” is recommended in “Microsoft's UEFI CA signing policy” document [2] and since it is known to Microsoft greatly simplifies and accelerates the review process.

From the original shim github description [1]:
“shim is a trivial EFI application that, when run, attempts to open and execute another application. It

will initially attempt to do this via the standard EFI LoadImage() and StartImage() calls. If these fail (because secure boot is enabled and the binary is not signed with an appropriate key, for instance) it will then validate the binary against a built-in certificate. If this succeeds and if the binary or signing key are not blacklisted then "shim" will relocate and execute the binary.

shim will also install a protocol which permits the second-stage bootloader to perform similar binary validation. This protocol has a GUID as described in the shim.h header file and provides a single entry point."

Shim may thus be used to boot a suitably signed EFI payload that runs OpenXT. We have provided a tool that parses a grub configuration boot stanza and builds a single EFI payload containing the specified configuration (typically tboot, xen, dom0 kernel and dom0 initrd images and associated command lines). This payload must be signed by a key that has its certificate embedded in shim.

Having verified the signature of the payload, shim will transfer control to it. The payload will extract/decompress the images into memory, synthesize a multiboot structure and associated machine information tables, then jump into the 32b entry point of the first image. If tboot is used, TXT will be used to perform a DRoT measurement of the payload components for use in addition to the SRoT measurements during the Secure Boot. tboot will then transfer control to Xen, which will create dom0 using the kernel and initrd images. The dom0 initrd is responsible for continuing the chain of trust on through the dom0 filesystem as per standard OpenXT practice of using the TPM to seal the dom0 filesystem AES XTS encryption key. It may be desirable to add additional SRoT PCRs (e.g. 0, 2, 4) to the set used to seal the key in addition to the DRoT PCR (17, 18, 19).

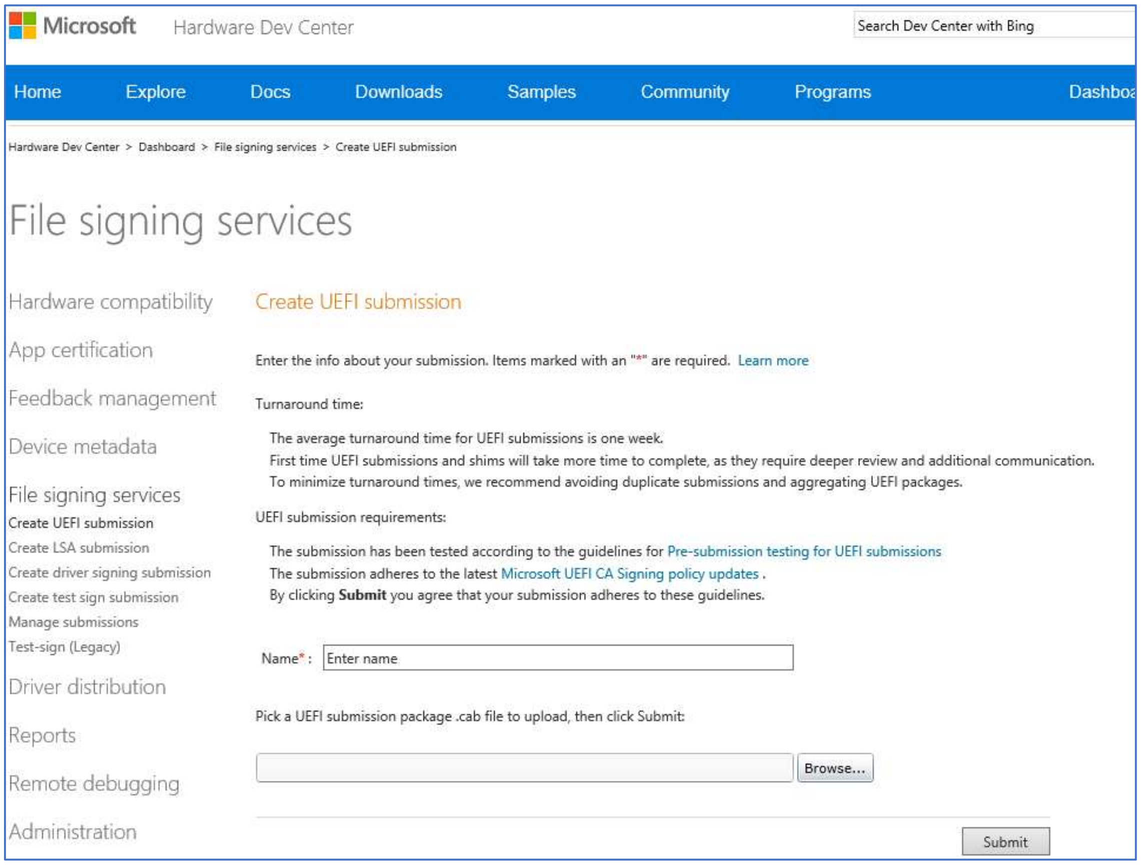
Microsoft require that the certificate built-in to shim be one that is held on a HSM, so a suitable device must be acquired for this purpose. The certificate must be extracted and embedded in the shim source code, and then submitted to Microsoft for signing. The HSM may then be used to sign updated payloads without further interaction with Microsoft.

4.2 Microsoft UEFI Signing Service

The Microsoft UEFI signing service requires two prerequisites:

- Extended Validation (EV) Code Signing Certificate
  EV is similar to standard code signing certificate except that it requires verification of the requesting entity's identity by a CA. Currently Microsoft recommended list of CAs includes: DigiCert, Entrust, GlobalSign, Symantec and Certum. More details in [3]
- Hardware Developer Program account
  Registration process requires EV certificate. Most recent description of the procedure in [4]

Hardware Developer Program account allows accessing SysDev dashboard [5] with Microsoft File signing service. Once logged-in to SysDev portal, proceed with "Create UEFI submission" [6] in "File signing services" tile. This should result in following UEFI submission page:



File signing services

Create UEFI submission

Enter the info about your submission. Items marked with an "*" are required. Learn more

Turnaround time:

The average turnaround time for UEFI submissions is one week.
First time UEFI submissions and shims will take more time to complete, as they require deeper review and additional communication.
To minimize turnaround times, we recommend avoiding duplicate submissions and aggregating UEFI packages.

UEFI submission requirements:

The submission has been tested according to the guidelines for Pre-submission testing for UEFI submissions
The submission adheres to the latest Microsoft UEFI CA Signing policy updates .
By clicking **Submit** you agree that your submission adheres to these guidelines.

Name* : Enter name

Pick a UEFI submission package .cab file to upload, then click Submit:

Name of the submission is required.

Submission package should be either .efi file or .efi file archived in a .cab file. .efi file should be standard UEFI PE binary singed with EV key registered with Hardware Developer Program account. EBC binaries are not accepted. List of requirements for UEFI signing submissions included in [2].

Submitting valid submission package should result in direct email contact with a member of Microsoft Windows Sysdev UEFI Signing Program. That part is not automated and usually requires filling out provided questionnaire and sending it back.

Questionnaire should be similar to the following list of questions (comments relevant to the project *[in-lined]*):

- Provide URLs to your company website and products included in this Submission.
- Does the UEFI submission contain final shipping product (RTM – Release to Manufacture), to better understand this question please go through signing policy updates published in SysDev blog?
  [UEFI signing policy requires UEFI submission to be production quality.]
- Have you tested your product following Pre-Submission testing document?
  [UEFI signing policy requires UEFI submission to be production quality.]
- Provide a description of the purpose and functions of your submitted modules.
  [It's a clone of the shim, suggested in "Microsoft's UEFI CA signing policy" document [2]*, that boots type-1 security hypervisor.]*
- Is this a version update to any of your earlier signed UEFI submission, if yes then:
  - What is the Submission ID of previous submission?

    - Is there a security fix involved in this newer version? If yes, provide brief explanation of fixes and impact
- Does the UEFI submission include license terms that would mandate the transmission of keys used for signing? (ie..,GPLv3 or similar)
  [No, it's a clone of the shim suggested in "Microsoft's UEFI CA signing policy" document [2]*, the license does not require the transmission of keys.]*
- Does your submission load and execute any other code prior to ExitBootServices()? If so, please explain how the code is loaded, authorized, and executed.
  [Yes, it does. Loaded binaries are checked against the embedded EV certificate in .vendor_cert section of the binary, and if they pass they are permitted to execute.
  The binaries loaded by the shim ensure the continuation of the chain of trust, by checking any other modules they may load using the shim protocol, or UEFI's BS->LoadImage.]
- Does your submission take any user input? And what are the input validation performed?
  [No.]
- Does your submission take any programmatic input (files on disc or UEFI variables etc.)? And what are the input validation performed?
  [It checks a revocation list of hashes and certificates, as required by Microsoft's UEFI CA signing policy document, the data are parsed carefully, and we have tested with various invalid and fuzzed input.]
- Have you done any security review of your product, internal or external?
  [Yes, internal.]
- Does your product use GRUB OR other loaders? If yes, can you describe where the source code originated and what Secure Boot protections have been enabled in it?
  [No, it doesn't]
- Do you use OpenSSL, if yes then what is the version?
  [Yes, current version OpenSSL 1.1.0e]
- Is it a EfiByteCode? If yes, then what are the platforms supported?
  [No.]
- Is it specific to any OEM specific device only? And if yes, which ones?
  [No.]
- Is this a SHIM? If yes then:
    - What is the origin of your SHIM and full version number?
      [It's a shim suggested in "UEFI CA signing policy document" document [2]*. Following version was cloned: https://github.com/rhinstaller/shim/commit/ea5f7e15971358b972b3a42656f316db588f5311]*
    - How do you manage and protect the keys used in your SHIM?
      [We use an HSM in a physically secure location, with two factor authentication.]
    - Do you use EV certificates as embedded certificates in SHIM?
      [Yes.]
    - If your SHIM launches GRUB or other loaders, what is the source code origin and full version?
      [It doesn't]
    - If your SHIM launches anything else, provide further details about what is launched and how it prevents execution of unauthenticated code
      [It launches our type-1 security hypervisor. Any binaries or modules that may be subsequently loaded are verified either using the shim protocol, or by using UEFI's BS-

>LoadImage both of which verify the binary. This guarantees the chain of trust is maintained.]

- Does it load GRUB or other loaders which support loading unsigned kernels?
  [No, the binary loaded contains all the ring 0 code for the system, and does not load anything else which isn't both signed and measured, nor will our signing policy permit us to sign anything that would.]
- What are the changes made to this compared to your last one signed (if this is not your first submission)?

References

[1] "shim," [Online]. Available: https://github.com/rhinstaller/shim.

[2] "UEFI CA Signing policy updates," [Online]. Available: https://blogs.msdn.microsoft.com/windows_hardware_certification/2013/12/03/microsoft-uefi-ca-signing-policy-updates.

[3] "Get a code signing certificate," [Online]. Available: https://docs.microsoft.com/en-gb/windows-hardware/drivers/dashboard/get-a-code-signing-certificate.

[4] "Register for the Hardware Program," [Online]. Available: https://docs.microsoft.com/en-gb/windows-hardware/drivers/dashboard/register-for-the-hardware-program.

[5] "SysDev dashboard," [Online]. Available: https://sysdev.microsoft.com.

[6] "UEFI submission," [Online]. Available: https://sysdev.microsoft.com/en-GB/Hardware/member/FileSigningServices/CreateUefiRequest.aspx .