Protecting system integrity with Trusted Platform Module

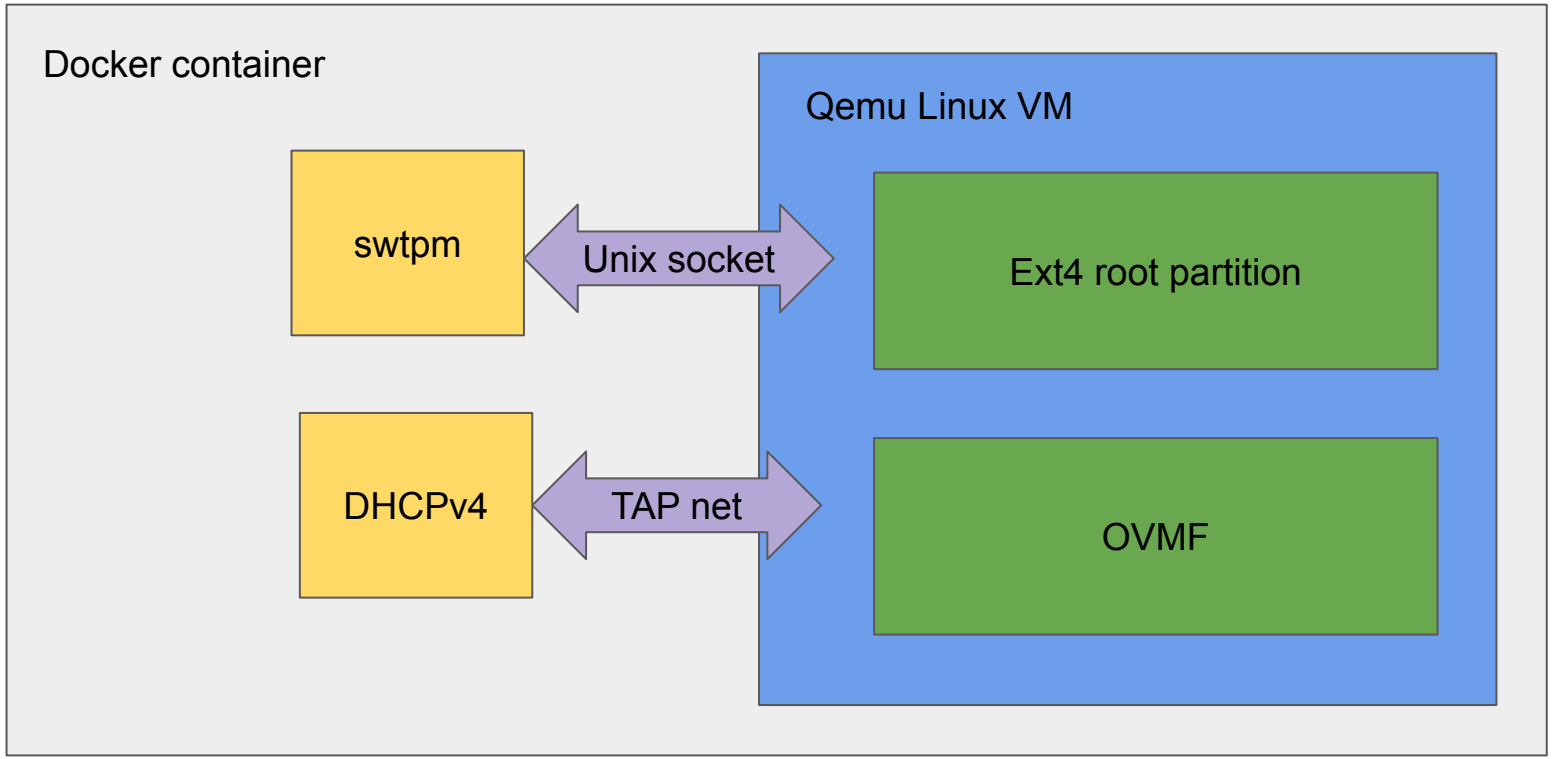
Dmitrii Potoskuev [dpotoskuev@fb.com]
Production Engineer

FACEBOOK

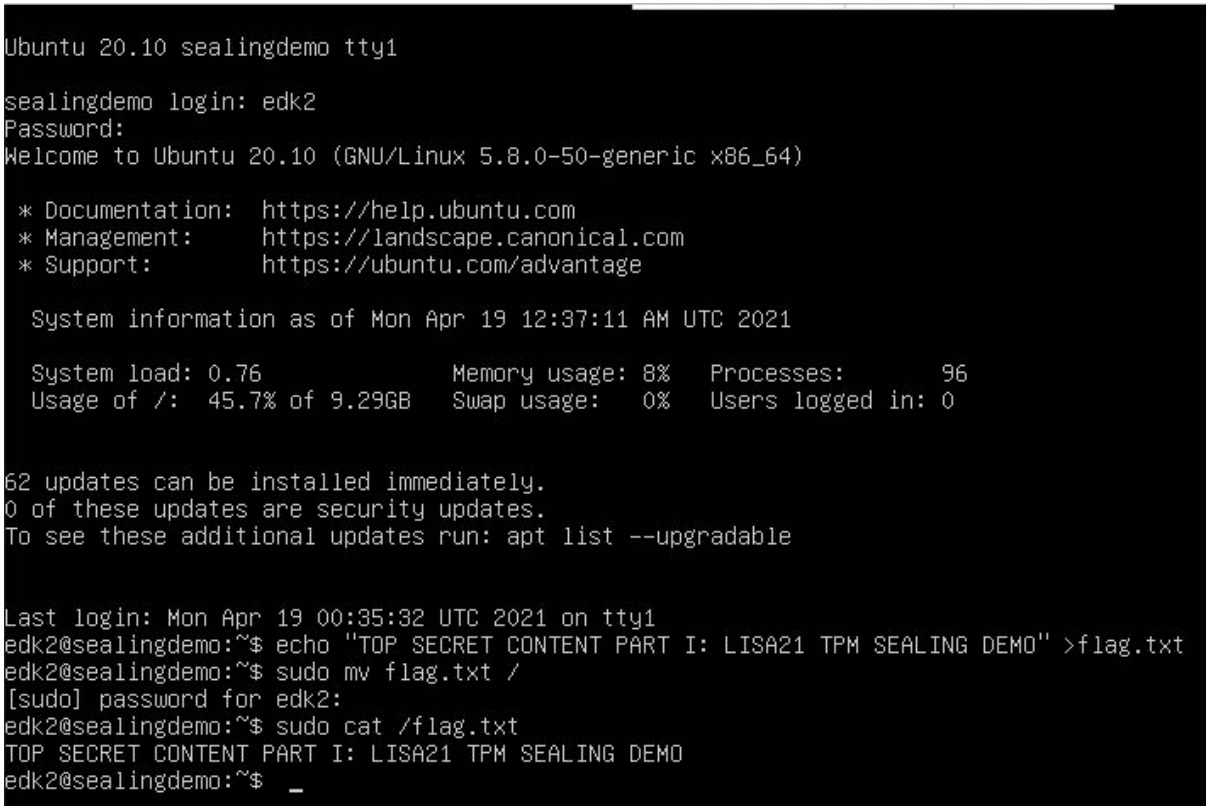
Agenda

- Setup description
- Compromising secret using demo DXE malicious driver
- Using measured boot and TPM sealing capabilities to prevent the attack
- Q&A

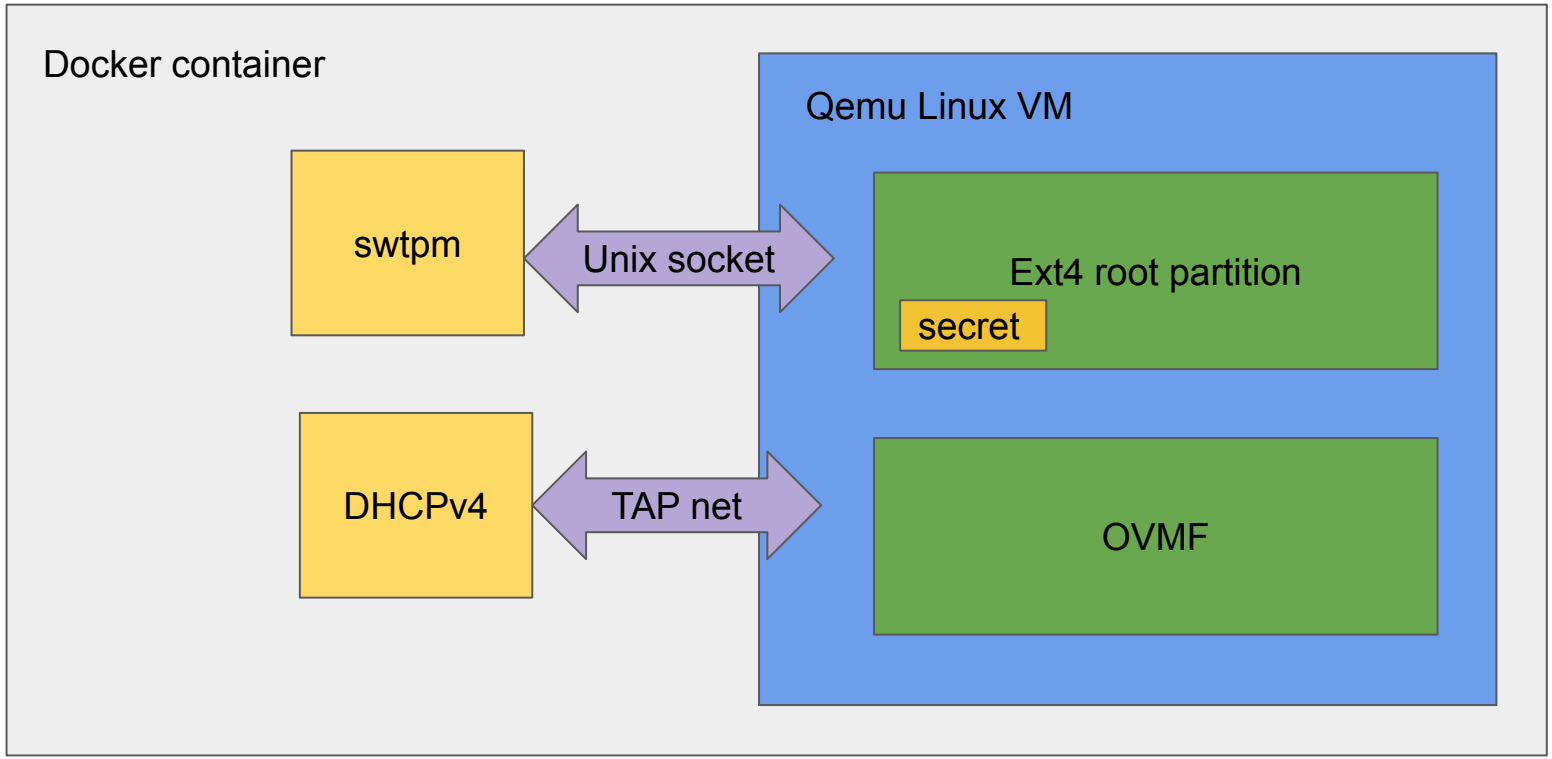
Setup


Docker container
Qemu Linux VM
swtpm
Unix socket
Ext4 root partition
DHCPv4
TAP net
OVMF

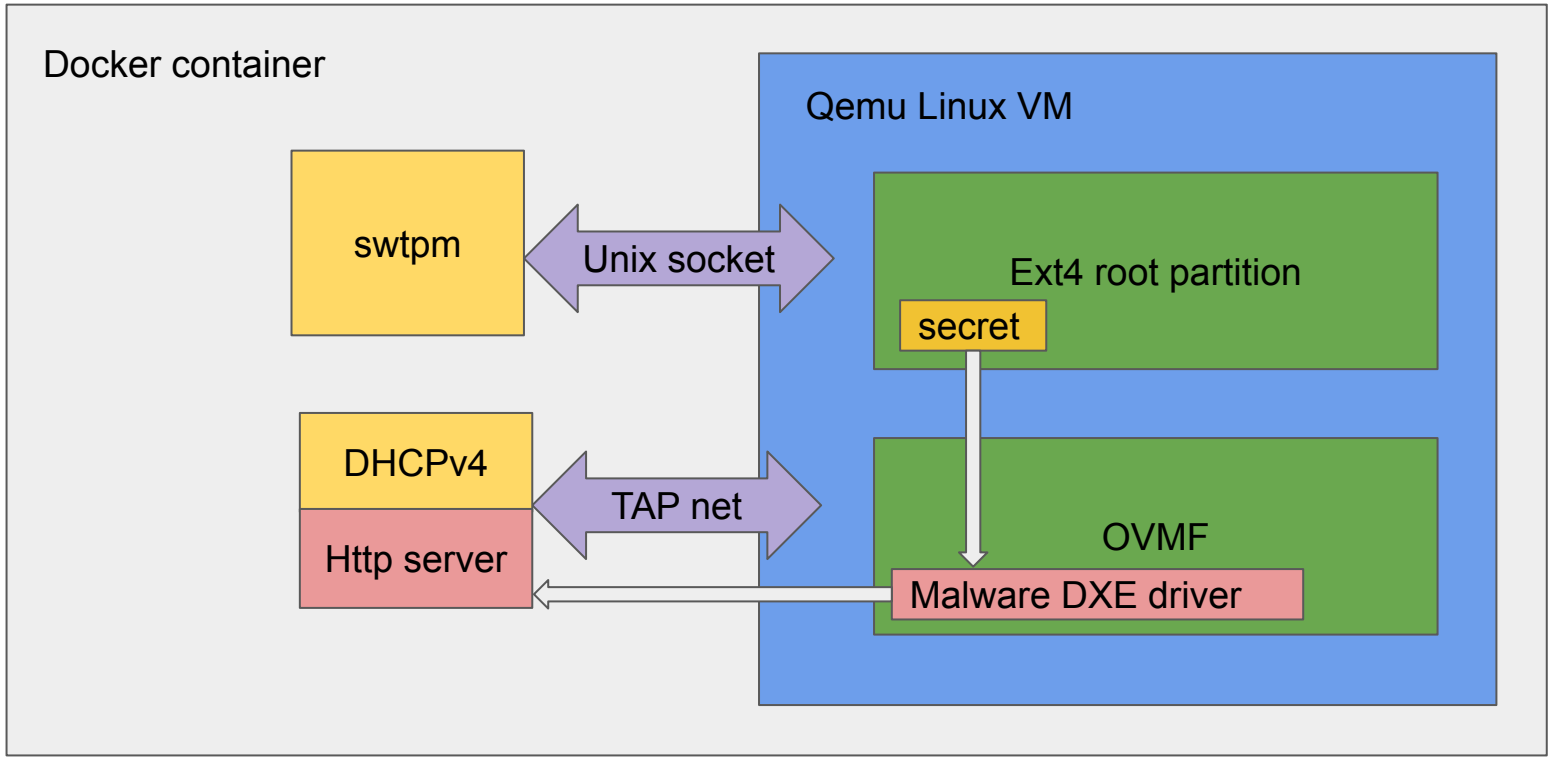
Creating a secret



Setup


Docker container
Qemu Linux VM
swtpm
Unix socket
Ext4 root partition
secret
DHCPv4
TAP net
OVMF

Setup


Docker container
Qemu Linux VM
swtpm
Unix socket
Ext4 root partition
secret
DHCPv4
Http server
TAP net
OVMF
Malware DXE driver

Malware at work

```
[FbM] Locating handle for FileSystem protocol
[FbM] Could not open FileSystem protocol on handle 0, trying next
[FbM] secret found:TOP SECRET CONTENT PART I: LISA21 TPM SEALING DEMO
[FbM] Running DHCPv4 2
[FbM] Dhcp4 starting
[FbM] Interface name is: eth0
[FbM] Sending the secret to external host
[FbM] HTTP Request successful, getting response
[FbM] ReadyToBoot done
BdsDxe: loading Boot0007 "ubuntu" from HD(1,GPT,DEA400E6-5F8A-4D34-AC83-B9BBAE3A
6065,0x800,0x100000)/\EFI\ubuntu\shimx64.efi
BdsDxe: starting Boot0007 "ubuntu" from HD(1,GPT,DEA400E6-5F8A-4D34-AC83-B9BBAE3
A6065,0x800,0x100000)/\EFI\ubuntu\shimx64.efi
```


```
edk2@677244790add:~$ python3 malware_server.py
Headers: content-length: 49


Data: TOP SECRET CONTENT PART I: LISA21 TPM SEALING DEMO
192.168.30.53 - - [19/Apr/2021 08:57:08] "POST / HTTP/1.1" 200 -
All done!
```

Example malicious DXE driver

- Created using EDK II
- ~ 700 LOC in C
- DHCPv4/v6 support
- Ext2/3/4 filesystem support
- ~ 16Kb compiled, and ~200Kb of support DXE drivers (libraries)
- Tested on real hardware

EDK II

- Open Source firmware development environment
- Among other things, allows to build UEFI DXE drivers
- Used to build our “demo malware”

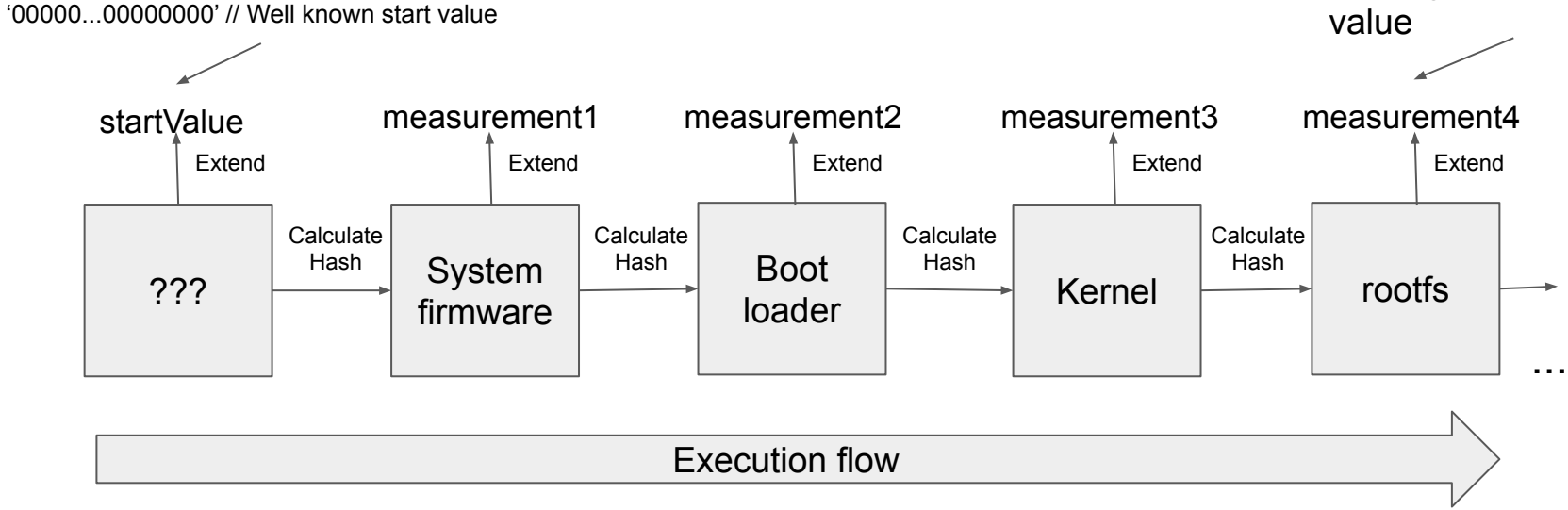
Could Secure Boot help?

- Yes, in some cases...
- Doesn’t scale well - flash memory bitflips and human errors can cause big problems
- How to make sure that it’s enabled and uses a right key?

Measured boot

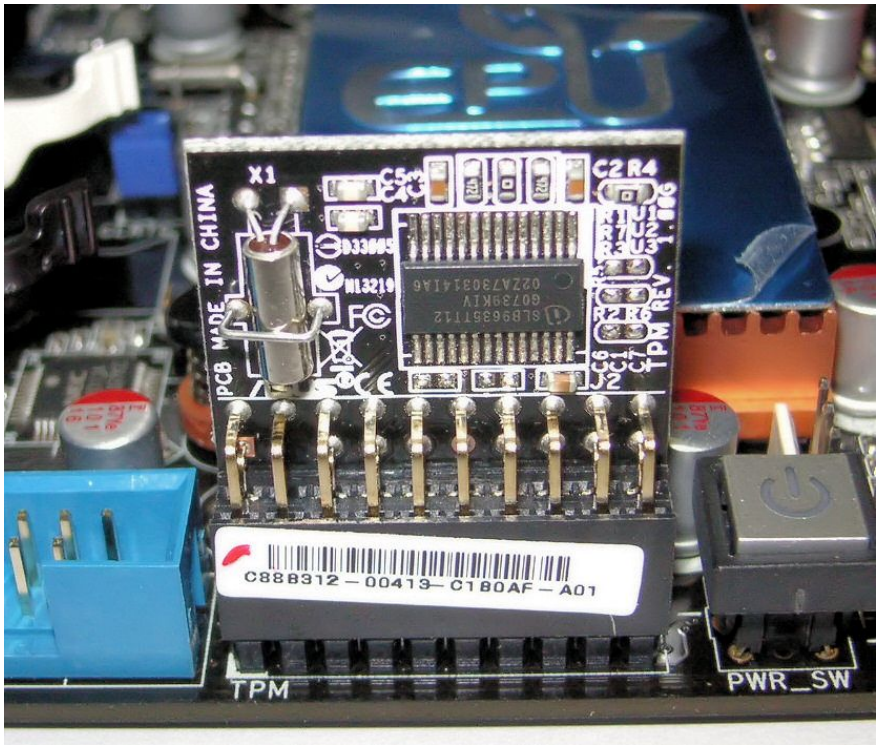
```
func extendHash(current_value, extension) {
    return SHA256(current_value | SHA256(extension))
}
```

The last measurement in the chain depends on all previous ones, starting from the initial value



Trusted Platform Module

- Hardware storage of measurements - Platform Configuration Registers (PCRs)
- Allows only “extension”, there is no way to change PCR values directly
- Is able to export and sign list of PCR values - create a “Quote”
- Has a unique identity which allows to check if the Quote came from a real chip



Measured boot with sealing approach

- Do remote attestation
- Deliver the secret to the system
- Seal the secret to the “authorize” policy
- Check that the system state during sealing matches the one after attestation
- Deliver a signed policy which allows the secret to be unsealed only in a known good state
- Use the signed policy to unseal the secret after system reboots

Measured boot with sealing approach

- Do remote attestation
- Deliver the secret to the system
- Seal the secret to the “authorize” policy
- Check that the system state during sealing matches the one after attestation
- Deliver a signed policy which allows the secret to be unsealed only in a known good state
- Use the signed policy to unseal the secret after system reboots

These we will demonstrate

Generation of policy signing key

```
edk2@sealingdemo:~/sealing_demo$ mkdir policy_signing_key
edk2@sealingdemo:~/sealing_demo$ cd policy_signing_key/
edk2@sealingdemo:~/sealing_demo/policy_signing_key$ openssl genrsa -out priv.pem 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.............................................+++++
..............+++++
e is 65537 (0x010001)
edk2@sealingdemo:~/sealing_demo/policy_signing_key$ openssl rsa -in priv.pem -out pub.pem -pubout
writing RSA key
edk2@sealingdemo:~/sealing_demo/policy_signing_key$ ls
priv.pem  pub.pem
edk2@sealingdemo:~/sealing_demo/policy_signing_key$ tpm2_loadexternal -G rsa -C o -u pub.pem -c key.ctx -n pub.name
name: 000bd3ad44ca46cff35a4c0ac7b5f0d3ec449525cfa5a8065676246294f10d5f5a0f
edk2@sealingdemo:~/sealing_demo/policy_signing_key$ ls
key.ctx  priv.pem  pub.name  pub.pem
edk2@sealingdemo:~/sealing_demo/policy_signing_key$ rm -f key.ctx
edk2@sealingdemo:~/sealing_demo/policy_signing_key$ _
```

Authorize policy creation

```
edk2@sealingdemo:~/sealing_demo/policy_signing_key$
edk2@sealingdemo:~/sealing_demo/policy_signing_key$ cd ..
edk2@sealingdemo:~/sealing_demo$ tpm2_startauthsession -S auth.ctx
edk2@sealingdemo:~/sealing_demo$ tpm2_policyauthorize -S auth.ctx -L authorized.policy -n ./policy_signing_key/pub.name
293b976e35f97ee6668aec36d9e74c4b73d34912962c92cb65a5acfb8b6dd14e
edk2@sealingdemo:~/sealing_demo$ tpm2_flushcontext auth.ctx
edk2@sealingdemo:~/sealing_demo$ rm -f auth.ctx
edk2@sealingdemo:~/sealing_demo$ ls
authorized.policy  policy_signing_key
edk2@sealingdemo:~/sealing_demo$
```

Sealing the secret

```
edk2@sealingdemo:~/sealing_demo$
edk2@sealingdemo:~/sealing_demo$ tpm2_createek -c ek.ctx
edk2@sealingdemo:~/sealing_demo$ tpm2_startauthsession --policy-session -S auth.ctx
edk2@sealingdemo:~/sealing_demo$ tpm2_policysecret -S auth.ctx -c e
837197674484b3f81a90cc8d46a5d724fd52d76e06520b64f2a1da1b331469aa
edk2@sealingdemo:~/sealing_demo$ echo "TOP SECRET CONTENT PART II: LISA21 TPM SEALING DEMO" | tpm2_create -g sha256 -u secret.pu
b -r secret.priv -i- -C ek.ctx -L authorized.policy -P "session:auth.ctx"
name-alg:
  value: sha256
  raw: 0xb
attributes:
  value: fixedtpm|fixedparent
  raw: 0x12
type:
  value: keyedhash
  raw: 0x8
algorithm:
  value: null
  raw: 0x10
keyedhash: 09b8ccae1d7a356af75dfc18a60223971c05c2da30ec3f4c2502a5478a560991
authorization policy: 293b976e35f97ee6668aec36d9e74c4b73d34912962c92cb65a5acfb8b6dd14e
edk2@sealingdemo:~/sealing_demo$ tpm2_flushcontext auth.ctx
edk2@sealingdemo:~/sealing_demo$ ls
auth.ctx  authorized.policy  ek.ctx  policy_signing_key  secret.priv  secret.pub
edk2@sealingdemo:~/sealing_demo$ rm -f auth.ctx ek.ctx authorized.policy
edk2@sealingdemo:~/sealing_demo$ ls
policy_signing_key  secret.priv  secret.pub
edk2@sealingdemo:~/sealing_demo$
```

Is it really encrypted? :)

```
edk2@sealingdemo:~/sealing_demo$
edk2@sealingdemo:~/sealing_demo$ xxd secret.priv
00000000: 00b2 0020 cef8 afff ac8c 5a11 9856 f9b6  ... ......Z..V..
00000010: fe50 b919 4d9b 0f80 d3ee 82e4 0d4c b9a2  .P..M........L..
00000020: a997 7313 0010 eff3 366e 7996 81c7 a0bc  ..s.....6ny.....
00000030: 3cc1 9972 c24f db83 6602 663f 569e 61ed  <..r.O..f.f?V.a.
00000040: 570c ac81 b6fd ceba c688 ec43 cb03 dc50  W..........C...P
00000050: 57af 1d9e f8b1 47f6 b0e6 b875 9098 5e13  W.....G....u..^.
00000060: cd1b d8e1 ca3c b889 1fa9 a14b c79b 79be  .....<.....K..y.
00000070: 0ede 15f5 1331 728c 3c80 ede8 43a6 e74c  .....1r.<...C..L
00000080: 98cd 0b05 fb9a dca2 7caf acb6 eb8a ef42  ........|......B
00000090: 1efe 5e58 ad08 fa8e 7418 9359 63c8 8e48  ..^X....t..Yc..H
000000a0: 261a ce1a ec8e 4027 60b8 dd4b 86ba c8e8  &.....@'`..K....
000000b0: d5e3 637b                                ..c{
edk2@sealingdemo:~/sealing_demo$ xxd secret.pub
00000000: 004e 0008 000b 0000 0012 0020 293b 976e  .N......... );.n
00000010: 35f9 7ee6 668a ec36 d9e7 4c4b 73d3 4912  5.~.f..6..LKs.I.
00000020: 962c 92cb 65a5 acfb 8b6d d14e 0010 0020  .,..e....m.N...
00000030: 09b8 ccae 1d7a 356a f75d fc18 a602 2397  .....z5j.]....#.
00000040: 1c05 c2da 30ec 3f4c 2502 a547 8a56 0991  ....0.?L%..G.V..
edk2@sealingdemo:~/sealing_demo$
edk2@sealingdemo:~/sealing_demo$ ls
policy_signing_key  secret.priv  secret.pub
edk2@sealingdemo:~/sealing_demo$
edk2@sealingdemo:~/sealing_demo$ _
```

Creating PCR policy

TODO

- Obtain PCR values, which match a desired system state.
- Generate the PCR policy
- Sign it with the policy signing key
- Deliver to the system

Creating PCR policy

```
edk2@sealingdemo:~/sealing_demo$
edk2@sealingdemo:~/sealing_demo$ tpm2_pcrread -o pcr.dat "sha1:0"
sha1:
  0 : 0x5564B9D251554FC4DB5F8B1D5C94F1360EA7E51F
edk2@sealingdemo:~/sealing_demo$ xxd pcr.dat
00000000: 5564 b9d2 5155 4fc4 db5f 8b1d 5c94 f136  Ud..QUO.._..\..6
00000010: 0ea7 e51f                                ....
edk2@sealingdemo:~/sealing_demo$ ls
pcr.dat  policy_signing_key  secret.priv  secret.pub
edk2@sealingdemo:~/sealing_demo$ tpm2_startauthsession -S auth.ctx
edk2@sealingdemo:~/sealing_demo$ tpm2_policypcr -S auth.ctx -l sha1:0 -f pcr.dat -L pcr0.sha1.policy
5571ed199aa5f459a4e324ea9795a8aaf41b4a8dda61d8f85b5706580b2a3b65
edk2@sealingdemo:~/sealing_demo$ tpm2_flushcontext auth.ctx
edk2@sealingdemo:~/sealing_demo$ openssl dgst -sha256 -sign ./policy_signing_key/priv.pem -out pcr_policy.signature pcr0.sha1.po
licy
edk2@sealingdemo:~/sealing_demo$ ls
auth.ctx  pcr0.sha1.policy  pcr.dat  pcr_policy.signature  policy_signing_key  secret.priv  secret.pub
edk2@sealingdemo:~/sealing_demo$ rm -f auth.ctx pcr.dat
edk2@sealingdemo:~/sealing_demo$
```

Unseal the secret

TODO

- Verify policy signature
- Satisfy PCR policy
- Satisfy Authorize policy
- Load the sealed secret to TPM
- Actually unseal the secret

Unseal the secret

Verify PCR policy signature

```
edk2@sealingdemo:~/sealing_demo$ tpm2_loadexternal -G rsa -C o -u ./policy_signing_key/pub.pem -c psk.ctx
name: 000bd3ad44ca46cff35a4c0ac7b5f0d3ec449525cfa5a8065676246294f10d5f5a0f
edk2@sealingdemo:~/sealing_demo$ tpm2_verifysignature -c psk.ctx -g sha256 -m pcr0.sha1.policy -s pcr_policy.signature -t verifi
cation.tkt -f rsassa
edk2@sealingdemo:~/sealing_demo$ ls
pcr0.sha1.policy  pcr_policy.signature  policy_signing_key  psk.ctx  secret.priv  secret.pub  verification.tkt
edk2@sealingdemo:~/sealing_demo$ rm -f psk.ctx
```

Satisfy PCR and authorize policies

```
edk2@sealingdemo:~/sealing_demo$ tpm2_startauthsession --policy-session -S auth.ctx
edk2@sealingdemo:~/sealing_demo$ tpm2_policypcr -S auth.ctx -l sha1:0
5571ed199aa5f459a4e324ea9795a8aaf41b4a8dda61d8f85b5706580b2a3b65
edk2@sealingdemo:~/sealing_demo$ tpm2_policyauthorize -S auth.ctx -i pcr0.sha1.policy -n ./policy_signing_key/pub.name -t verifi
cation.tkt
293b976e35f97ee6668aec36d9e74c4b73d34912962c92cb65a5acfb8b6dd14e
edk2@sealingdemo:~/sealing_demo$ ls
auth.ctx  pcr0.sha1.policy  pcr_policy.signature  policy_signing_key  secret.priv  secret.pub  verification.tkt
edk2@sealingdemo:~/sealing_demo$
```

Unseal the secret

Load sealed secret to TPM

```
edk2@sealingdemo:~/sealing_demo$ tpm2_createek -c ek.ctx
edk2@sealingdemo:~/sealing_demo$ tpm2_startauthsession --policy-session -S ek_auth.ctx
edk2@sealingdemo:~/sealing_demo$ tpm2_policysecret -S ek_auth.ctx -c e
837197674484b3f81a90cc8d46a5d724fd52d76e06520b64f2a1da1b331469aa
edk2@sealingdemo:~/sealing_demo$ tpm2_load -C ek.ctx -u secret.pub -r secret.priv -c secret.ctx -P session:ek_auth.ctx
name: 000b9ccadd97edc34614378b3523b411c6cbca0e5a0b37d14bc3068c547bf78b4f3a
edk2@sealingdemo:~/sealing_demo$ tpm2_flushcontext ek_auth.ctx
edk2@sealingdemo:~/sealing_demo$ ls
auth.ctx     ek.ctx           pcr_policy.signature  secret.ctx   secret.pub
ek_auth.ctx  pcr0.sha1.policy  policy_signing_key    secret.priv  verification.tkt
edk2@sealingdemo:~/sealing_demo$ rm -f ek_auth.ctx ek.ctx
```

Unseal the secret

```
edk2@sealingdemo:~/sealing_demo$
edk2@sealingdemo:~/sealing_demo$ tpm2_unseal -p session:auth.ctx -c secret.ctx
TOP SECRET CONTENT PART II: LISA21 TPM SEALING DEMO
edk2@sealingdemo:~/sealing_demo$ tpm2_flushcontext auth.ctx
edk2@sealingdemo:~/sealing_demo$ ls
auth.ctx  pcr0.sha1.policy  pcr_policy.signature  policy_signing_key  secret.ctx  secret.priv  secret.pub  verification.tkt
edk2@sealingdemo:~/sealing_demo$ rm -f auth.ctx secret.ctx
edk2@sealingdemo:~/sealing_demo$ _
```

Boot with malicious firmware and try to satisfy policies

```
edk2@sealingdemo:~/sealing_demo$ tpm2_pcrread sha1:0
sha1:
  0 : 0x19DE21F1C2844E4FDC180288D85384392A40FD74
edk2@sealingdemo:~/sealing_demo$ tpm2_startauthsession --policy-session -S auth.ctx
edk2@sealingdemo:~/sealing_demo$ tpm2_policypcr -S auth.ctx -l sha1:0
d8cb2e28ec827e5dc249edcc234e8529f1aae532acc869961cb4b03c98ec6062
edk2@sealingdemo:~/sealing_demo$ tpm2_policyauthorize -S auth.ctx -i pcr0.sha1.policy -n ./policy_signing_key/pub.name -t verifi
cation.tkt
WARNING:esys:src/tss2-esys/api/Esys_PolicyAuthorize.c:306:Esys_PolicyAuthorize_Finish() Received TPM Error
ERROR:esys:src/tss2-esys/api/Esys_PolicyAuthorize.c:108:Esys_PolicyAuthorize() Esys Finish ErrorCode (0x000001c4)
ERROR: Esys_PolicyAuthorize(0x1C4) - tpm:parameter(1):value is out of range or is not correct for the context
ERROR: Could not build tpm authorized policy
ERROR: Unable to run tpm2_policyauthorize
edk2@sealingdemo:~/sealing_demo$ tpm2_flushcontext auth.ctx
edk2@sealingdemo:~/sealing_demo$ rm -f auth.ctx
edk2@sealingdemo:~/sealing_demo$
```

```
edk2@sealingdemo:~/sealing_demo$
edk2@sealingdemo:~/sealing_demo$ tpm2_pcrread -o pcr.dat "sha1:0"
sha1:
  0 : 0x5564B9D251554FC4DB5F8B1D5C94F1360EA7E51F
edk2@sealingdemo:~/sealing_demo$ xxd pcr.dat
00000000: 5564 b9d2 5155 4fc4 db5f 8b1d 5c94 f136  Ud..QUO.._..\..6
00000010: 0ea7 e51f                                ....
edk2@sealingdemo:~/sealing_demo$ ls
pcr.dat  policy_signing_key  secret.priv  secret.pub
edk2@sealingdemo:~/sealing_demo$ tpm2_startauthsession -S auth.ctx
edk2@sealingdemo:~/sealing_demo$ tpm2_policypcr -S auth.ctx -l sha1:0 -f pcr.dat -L pcr0.sha1.policy
5571ed199aa5f459a4e324ea9795a8aaf41b4a8dda61d8f85b5706580b2a3b65
```

Questions time

Happy sealing! :)

Dmitrii Potoskuev [dpotoskuev@fb.com]