Secure Hibernation in a Lockdown World

Matthew Garrett <matthewg@nvidia.com>

Lockdown recap

- Strengthen the boundary between kernel and root
- Block code paths that would allow modification of the running kernel
- Used in various cases, but mostly with secure boot

Secure boot

- Verify that each component in the boot chain has a trusted signature
- Refuse to boot components that don’t
- UEFI, embedded platforms

Hibernation

- Stores contents of RAM to disk
- Resume copies it back
- Resumes to the stored kernel, not the booted kernel

What protects the hibernation image?

- Nothing
- Attacker can write a hibernation image to disk and it will resume on next boot
- Attacker can use this to bypass secure boot

Protecting the data

- Sign or encrypt it
- But with what?
- What prevents the attacker providing a key?

TPMs

- Small hardware modules for cryptography
- Can generate keys while retaining the private half
- So we just generate a key on the TPM, right?

What prevents the attacker from doing the same?

- Attacker can ask the TPM to generate a key and then sign the image with it
- We need to be able to distinguish between legitimate and illegitimate keys

Platform Configuration Registers

- PCRs store data pushed to the TPM during boot
- Keys can include the PCR state during generation
- We can generate a key and then change the PCR
- Do this during boot and prove the key was generated by the kernel

But what stops an attacker using the key?

- Nothing
- We can’t keep it secret in a meaningful way

Controlling key usage

- Keys can have a usage policy that requires specific PCR state
- Have the kernel control a PCR and have policy tied to it
- Prove key was generated by kernel and can only be used by kernel

What stops an attacker doing the same?

- Old kernels won’t restrict the TPM
- Attacker can boot an old kernel, modify PCR state, generate a key and use it
- Need a way to differentiate new and old kernels

This gets ugly

- Have new kernels modify PCRs in a way the attacker can’t
- Requires a change in TPM state that occurs before users can run code
- But we don’t currently demark the handoff between kernel and userland

No, really

- During boot, the kernel EFI boot stub calls ExitBootServices()
- This modifies PCR 5
- We can extend PCR 5 before ExitBootServices() is called

Putting it together

- Modify PCR 5 during boot
- Restrict access to PCR 23
- Verify the key was generated by kernel and can only be used in-kernel
- Sign image on suspend, verify on resume, profit

But

- People are already using PCR 23
- Bother

NVIndexes

- TPM 2 specification allows creation of additional “PCRs” in TPM NVRam
- We can allocate one to the kernel and use it
- Allocated in a different range, minimal risk of collision

Oh no

- NV “PCRs” aren’t included in key creation data
- Can’t prove the key was generated by the kernel
- Ugh

Hurrah?

- TPM Audit Sessions
- TPM generates a running hash of each command executed and the reply
- TPM signs that hash on request

So…

- Start an audit session
- Read PCR 5 and NV PCR
- Generate the key
- Read PCR 5 and NV PCR again to avoid races
- Obtain signed digest of commands

What is the digest signed with?

- Attestation key
- How do we secure that?
- We don’t have to! Can only be used to sign TPM generated data
- Generate the AK on demand using a predictable seed, prove provenance of signature

What’s needed?

- Extend PCR 5 in boot stub
- Add in-kernel support for NVIndex PCRs
- Add audit session support
- Generate key within audit session
- Encrypt image with key
- Save key and session data/signature

And resume?

- Read key and audit data
- Verify audit signature
- Verify audit data contains expected PCR 5 and NV values
- Verify audit data corresponds to key
- Decrypt image
- Resume

Current status

- Implemented but hacky
- Will post cleaned-up patches shortly

Why not just check if swap is encrypted?

- Encryption is set up by initramfs
- Initramfs isn’t signed
- Attacker provides initramfs that sets up encrypted swap with known key

Risks

- This is somewhat convoluted
- Also requires strict access to TPM (Google abandoned this approach for this reason)

Further uses

- Modify NV “PCR” based on running app/container/whatever
- Have containers be able to prove their identity
- Sounds useful, is it?

Questions