TPM 1.2 key authorization values vulnerable to TPM transport eavesdropper

Low chrisfenner published GHSA-5x29-3hr9-6wpw on Jul 23, 2020

Package

go-tpm (Go)

Affected versions	Patched versions
< 0.3.0	0.3.0

Description

Impact

TPM 2.0 users are unaffected by this issue.

An adversary eavesdropping on the TPM 1.2 transport path can calculate `usageAuth` for a key created with CreateWrapKey, even though this value is encrypted as part of the TPM 1.2 command protocol.
The TPM 1.2 CreateWrapKey command accepts two secrets: `usageAuth` and `migrationAuth` . The ADIP protocol (TPM 1.2 specification, part 1, section 13.4) calls for these values to be encrypted with two different XOR keys.
Due to a bug in go-tpm prior to version 0.3.0, both `usageAuth` and `migrationAuth` are encrypted with the same XOR keystream. This allows an adversary to XOR `encUsageAuth` and `encMigrationAuth` together to calculate `usageAuth ^ encMigrationAuth` . Since `migrationAuth` is moot for all keys created with go-tpm's `CreateWrapKey` (since all keys created with this function are marked non-migratable), an adversary may guess or know (from code/binary inspection) that `migrationAuth` is all 0x00 bytes or some other fixed value. Such an adversary can then calculate `usageAuth` and use this value later to improperly use the created key, unbeknownst to the creator of the key.

Patches

Fixed in go-tpm version 0.3.0.

Workarounds

- TPM 2.0 users: No workaround needed. This issue only affects TPM 1.2 users.
- TPM 1.2 users: Call CreateWrapKey with a random 20-byte value for `migrationAuth` , even though that value is not used again (since CreateWrapKey creates keys that are non-migratable). Do not store or log this value.

Details

TPM 1.2 uses a protocol called ADIP (Authorization Data Insertion Protocol) to encrypt authorization values over-the-wire for newly created objects. This prevents a bus-snooping attack like those publicized by TPM Genie. TPM 2.0 makes this optional (the way to do it is with parameter-encryption sessions). You can read more about ADIP in section 13.4 of Part 1: Design Principles in the latest TPM 1.2 specification. Normally, ADIP consists of the following steps:

```
Key := SHA1(authSession.SharedSecret || a nonce)
Note: nonces and auth values in TPM 1.2 are always 20 bytes
EncAuth := XOR(Key, Auth)
```

When commands require one ADIP-encrypted auth value, the nonce is the last nonceEven (last nonce from the TPM).
When commands require two ADIP-encrypted auth values, the nonce for the first auth value is still nonceEven, and the nonce for the second auth value is the last nonceOdd, which is the one being provided by the caller along with the current command on the session.
The reason for this is that you wouldn't want an adversary to be able to XOR the two encrypted auth values together and come up with (auth 1 XOR key) XOR (auth 2 XOR key) where the "one-time" pad key is used twice and cancels itself out.

Here are the commands that take one authorization value by ADIP:

- Seal (the sealed data's auth value)
- Sealx (the sealed data's auth value)
- CreateKey (the key's auth value)
- MakeIdentity (the AIK's auth value)
- ChangeAuth (the entity's new auth value)
- ChangeAuthOwner (the new owner auth value)
- Delegate_CreateKeyDelegation (the new delegation auth value)
- Delegate_CreateOwnerDelegation (the new delegation auth value)
- NV_DefineSpace (the NV's auth value)
- CreateCounter (the counter's auth value)

Here are the commands that take two authorization values by ADIP:

- CreateWrapKey (the key's auth value, and the key's migration (to export out of the TPM) auth value)

The migrationAuth value is never used if the key does not have the TPM_KEY_FLAGS.migratable flag set on it. go-tpm does not currently allow the caller to set this flag.
Here was the bug in our implementation of CreateWrapKey():

go-tpm/tpm/tpm.go
Lines 1322 to 1329 in 16766ac

```
1322	var encUsageAuth digest
1323	for i := range srkAuth {
1324		encUsageAuth[i] = encAuthDataKey[i] ^ usageAuth[i]
1325	}
1326	var encMigrationAuth digest
1327	for i := range srkAuth {
1328		encMigrationAuth[i] = encAuthDataKey[i] ^ migrationAuth[i]
1329	}
```

Here we see that both usageAuth and migrationAuth are encrypted by the same XOR key. This is the correct key (i.e., it is based on nonceEven) for usageAuth, but not migrationAuth.

This means 2 things:

First: migrationAuth is being set to some value that is effectively unrelated to migrationAuth as passed by the caller. Again, this is not interesting to all current callers (given that there is no way for them to pass TPM_KEY_FLAGS.migratable via the current API; migrationAuth is not a meaningful value).

Second, and much more importantly: a user of go-tpm is vulnerable to the following attack by a passive bus-snooping adversary (CVE-2020-8918)
Wait for a CreateWrapKey command
Collect encUsageAuth and encMigrationAuth
Calculate (usageAuth XOR migrationAuth) := (encUsageAuth XOR encMigrationAuth)
Assuming migrationAuth is all 0x00 (a reasonable assumption for a caller who knows the key is not migratable), the calculation in (3) is the usage auth of the key.

Severity

Low

CVE ID

CVE-2020-8918

Weaknesses

No CWEs

Credits

chrisfenner