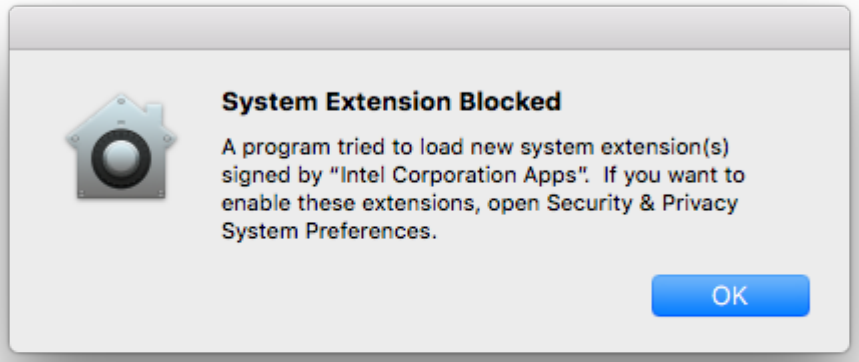
User Approved Kernel Extension Loading…

Published: 2017-08-29 · Archived: 2026-04-02 11:37:47 UTC

Apple is trying to improve security on the Mac, and starting with macOS High Sierra, kernel extensions that are installed with or after the installation of macOS High Sierra, will require user consent in order to load signed kernel extensions. This new feature should also make us more aware about the kernel extensions that we have installed. Some of which we may no longer need. After all. Not every uninstaller does what it should do.

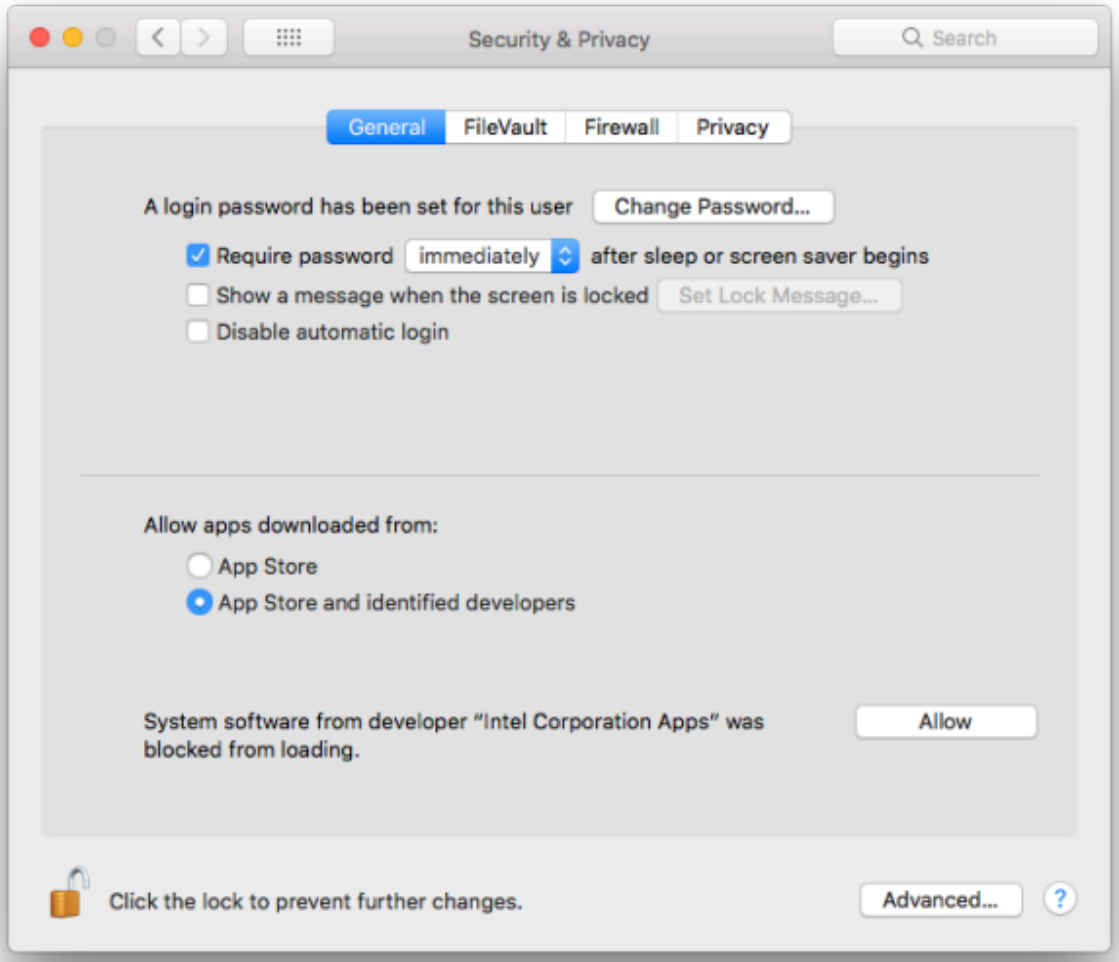
Ok. Let me start with an example:



This is what I got when I tried to load the Intel kernel extension (with sudo kextload EnergyDriver.kext) from a terminal window.

Please note that there is *nothing* wrong with this extension. It simply wasn't there, because I did a fresh installation of macOS High Sierra (a Developer Beta) and thus I had to allow it.

Anyway. Let's have a look at the updated Security and Privacy preference panel, with a new button to allow blocked kernel extension. Here it is:



It shows you the name of the developer. In this example it's one from 'Intel Corporation Apps'. And when you 'Allow' a blocked kernel extension, then the entry in the SQL database (KextPolicy) for this developer (per TeamID) will be updated accordantly.

Ok. Everything so far should be clear. Oh. One more thing. This feature is also known as User Approved Kernel Extension Loading. And *any* user can approve a kernel extension, even one *without* administrator privileges. Wait. What? Yup. Read this:

Kernel extensions don't require authorization if they:

> Were on the Mac before the upgrade to macOS High Sierra.
> Are replacing previously approved extensions.

This also explains why I had to allow the Intel kernel extension. Because it wasn't there before, and I had not approved it already. You may now wonder if we can we disable this new feature. No worries. You can. Here's what Apple has to say about disabling this feature:

If you want to disable User Approved Kernel Extension Loading, boot into macOS Recovery and use the spctl command. Run the command by itself to get more information about how to use the spctl command.

That's it. You're on your own. There is no detailed information about what you can, should and should not (try to) do. Great.

Let's start with the output of it

```
 Apple-iMacPro:~ Apple$ spctl

System Policy Basic Usage:
     spctl --assess [--type type] [-v] path ... # assessment
     spctl --add [--type type] [--path|--requirement|--anchor|--hash] spec ... # add rule(s)
     spctl [--enable|--disable|--remove] [--type type] [--path|--requirement|--anchor|--hash|--rule]
     spctl --status | --master-enable | --master-disable # system master switch

Kernel Extension User Consent Usage:
     spctl kext-consent ** Modifications only available in Recovery OS **
     status
          Print whether kernel extension user consent is enabled or disabled.
     enable
          Enable requiring user consent for kernel extensions.
     disable
          Disable requiring user consent for kernel extensions.
     add
          Insert a new Team Identifier into the list allowed to load kernel extensions without user
     list
          Print the list of Team Identifiers allowed to load without user consent.
     remove
          Remove a Team Identifier from the list allowed to load kernel extensions without user cons
```

```
Apple-iMacPro:~ Apple$ spctl kext-consent status
```

Kernel Extension User Consent: ENABLED

```
Apple-iMacPro:~ Apple$ spctl kext-consent disable
```

spctl: failed to store new configuration.

```
Apple-iMacPro:~ Apple$ sudo spctl kext-consent disable
```

Kernel Extension User Consent: DISABLED
Please restart for changes to take effect.

```
Apple-iMacPro:~ Apple$ sudo spctl kext-consent enable
```

Kernel Extension User Consent: ENABLED
Please restart for changes to take effect.

```
Apple-iMacPro:~ Apple$ spctl kext-consent list
```

spctl: no kext consent configuration found.

```
Apple-iMacPro:~ Apple$ sudo spctl kext-consent add 0123456789
Apple-iMacPro:~ Apple$ spctl kext-consent list
```

Allowed Team Identifiers:
0123456789

```
Apple-iMacPro:~ Apple$ sudo spctl kext-consent remove 0123456789
Apple-iMacPro:~ Apple$ spctl kext-consent list
```

Locating Your Team ID

The Team ID is a unique 10+ character string generated by Apple that's assigned to your team. You can find your Team ID using your developer account.

To locate your Team ID

Sign in to developer.apple.com/account and click Membership in the sidebar. Your Team ID appears in the Membership Information section under the team name.

NVRAM data
There are two properties that I would like to mention here. The first one being csr-active-config and the second one being csr-data. Let's first have a look at a snippet from the output of

```
Apple-iMacPro:~ Apple$ nvram -xp
```

```
1  < key >csr-active-config</ key >
2  < data >
3  gAIAAA==
4  </ data >
5  < key >csr-data</ key >
```

```
6  <data>
7  PGRpY3Q+PGtleT5rZXh0LWFsbG93ZWQtdGVhbXM8L2tleT48YXJyYXk+PHN0cmluZz4w
8  MTIzNDU2Nzg5PC9zdHJpbmc+PC9hcnJheT48L2RpY3Q+AA==
9  </data>
```

```
Apple-iMacPro:~ Apple$ echo -n gAIAAA==|base64 --decode|xxd -ps
```

0x80020000 (feature disabled)

```
Apple-iMacPro:~ Apple$ echo -n gAAAAA==|base64 --decode|xxd -ps
```

0x80000000 (feature enabled)

Note: You can use my csrstat command line tool to check the status of all SIP settings!

The base64 data of csr-data property is a dictionary with an array of all allowed team identifiers. Here is the one from our example

```
1  <dict>
2      <key>kext-allowed-teams</key>
3      <array>
4          <string>0123456789</string>
5      </array>
6  </dict>
```

And the csr-data property will still be there when you removed all team identifiers, but then with an empty array. Like this.

```
1  <dict>
2      <key>kext-allowed-teams</key>
3      <array>
4      </array>
```

5	`</ dict >`

Reset NVRAM will revert to its default state with User Approved Kernel Extension Loading enabled.

Enrolling in Mobile Device Management (MDM) automatically disables User Approved Kernel Extension Loading. The behavior for loading kernel extensions will be the same as macOS Sierra.

A future update to macOS High Sierra will allow you to use MDM to enable or disable User Approved Kernel Extension Loading, and to manage the list of kernel extensions which are allowed to load without user consent. If that is before or after the official release of macOS High Sierra is not known.

But you know what. All extensions in the prelinkedkernel are stripped. There is no data of the signature. Apple (still) relies on tools to check the kexts signature, before inclusion in the prelinkedkernel, and somehow, that doesn't feel right.

Update: The configuration is stored in: /var/db/SystemPolicyConfiguration/KextPolicy. I also found the SQL statements that spctl and systempolicyd are using to initialise and update the database:

```
DROP TABLE IF EXISTS kext_load_history

DROP TABLE IF EXISTS kext_load_history_v2

DROP TABLE IF EXISTS policy_by_team

DROP TABLE IF EXISTS policy_by_team_v2

DROP TABLE IF EXISTS policy_by_cdhash

DROP TABLE IF EXISTS policy_by_cdhash_v2

DELETE FROM kext_policy WHERE team_id = ?1 AND bundle_id = ?2

DELETE FROM kext_load_history_v3 WHERE team_id = ?1 AND bundle_id = ?2

SELECT bundle_id, developer_name FROM kext_policy WHERE team_id IS NULL

UPDATE kext_policy SET developer_name = ?1 WHERE team_id IS NULL and bundle_id = ?2

CREATE TABLE IF NOT EXISTS kext_load_history_v3 ( path TEXT PRIMARY KEY, team_id TEXT, bundle_id TEX

CREATE TABLE IF NOT EXISTS kext_policy ( team_id TEXT, bundle_id TEXT, allowed BOOLEAN, developer_na

SELECT allowed, flags FROM kext_policy WHERE team_id = ?1 AND bundle_id = ?2

SELECT allowed, flags FROM kext_policy WHERE team_id IS NULL AND bundle_id = ?2
```

```
SELECT allowed FROM kext_policy WHERE team_id = ?1", 0

INSERT INTO kext_policy (team_id, bundle_id, allowed, developer_name, flags) VALUES (?1, ?2, ?3, ?4,

UPDATE kext_policy SET allowed = ?3, flags = ((flags & ~?4) | ?5) WHERE team_id = ?1 AND bundle_id =

UPDATE kext_policy SET allowed = ?3, flags = ((flags & ~?4) | ?5) WHERE team_id IS NULL AND bundle_i

UPDATE kext_policy SET flags = (flags | ?3) WHERE team_id = ?1

UPDATE kext_policy SET flags = (flags | ?3) WHERE team_id IS NULL AND bundle_id = ?2

SELECT 1 FROM kext_load_history_v3 WHERE path = ?1

SELECT flags FROM kext_load_history_v3 WHERE path = ?1

UPDATE kext_load_history_v3 SET team_id = ?2, bundle_id = ?3, flags = ?4, last_seen = datetime('now'

INSERT INTO kext_load_history_v3 (path, team_id, bundle_id, boot_uuid, created_at, last_seen, flags)

SELECT created_at FROM kext_load_history_v3 WHERE team_id = ?1 AND bundle_id = ?2

SELECT created_at FROM kext_load_history_v3 WHERE team_id IS NULL AND bundle_id = ?2

SELECT last_seen FROM kext_load_history_v3 WHERE team_id = ?1 AND bundle_id = ?2

SELECT last_seen FROM kext_load_history_v3 WHERE team_id IS NULL AND bundle_id = ?2

SELECT 1 FROM kext_load_history_v3 WHERE team_id = ?1 AND bundle_id = ?2 AND (flags & ?3) = ?3 AND b

SELECT 1 FROM kext_load_history_v3 WHERE team_id IS NULL AND bundle_id = ?2 AND (flags & ?3) = ?3 AN

SELECT DISTINCT team_id FROM kext_policy WHERE team_id IS NOT NULL

SELECT bundle_id, developer_name, allowed FROM kext_policy WHERE team_id = ?1

SELECT bundle_id, developer_name, allowed FROM kext_policy WHERE team_id IS NULL

SELECT path, flags FROM kext_load_history_v3 WHERE team_id = ?1 AND bundle_id = ?2

SELECT path, flags FROM kext_load_history_v3 WHERE team_id IS NULL AND bundle_id = ?2

SELECT 1 FROM kext_policy WHERE flags & ?1 = ?1

UPDATE kext_policy SET allowed = 0, flags = ((flags & ~?1) | ?2) WHERE allowed = 1
```

You can use the above SQL statements with either /usr/bin/sqlite3 or some GUI app. Here is an example of the KextPolicy database:

path	bundle_id	team_id	boot_uuid	created_at	last_seen	
/Library/Extensions/ACS6x.kext	com.Accusys.driver.Acxxx	K3TDMD9Y6B	D2E6A228...	2017-08-31 10:58:46	2017-08-31 10:59:06	3
/Library/Extensions/ArcMSR.kext	com.Areca.ArcMSR	34JN824YNC	D2E6A228...	2017-08-31 10:58:46	2017-08-31 10:59:06	3
/Library/Extensions/ATTOCelerityFC8.kext	com.ATTO.driver.ATTOCelerityFC8	FC94733TZD	D2E6A228...	2017-08-31 10:58:46	2017-08-31 10:59:06	3
/Library/Extensions/ATTOExpressSASHBA2.kext	com.ATTO.driver.ATTOExpressSASHBA2	FC94733TZD	D2E6A228...	2017-08-31 10:58:46	2017-08-31 10:59:06	3
/Library/Extensions/ATTOExpressSASRAID2.kext	com.ATTO.driver.ATTOExpressSASRAID2	FC94733TZD	D2E6A228...	2017-08-31 10:58:46	2017-08-31 10:59:06	3
/Library/Extensions/CIJUSBLoad.kext	jp.co.canon.ij.print.CIJUSBLoad	XE2XNRRXZ5	D2E6A228...	2017-08-31 10:58:46	2017-08-31 10:59:03	9
/Library/Extensions/BJUSBLoad.kext	jp.co.canon.bj.print.BJUSBLoad	XE2XNRRXZ5	D2E6A228...	2017-08-31 10:58:46	2017-08-31 10:59:03	1
/Library/Extensions/CalDigitHDProDrv.kext	com.CalDigit.driver.HDPro	8R7PS6VYW7	D2E6A228...	2017-08-31 10:58:46	2017-08-31 10:59:06	3
/Library/Extensions/HighPointIOP.kext	com.highpoint-tech.kext.HighPointIOP	DX6G69M9N2	D2E6A228...	2017-08-31 10:58:46	2017-08-31 10:59:06	3
/Library/Extensions/HighPointRR.kext	com.highpoint-tech.kext.HighPointRR	DX6G69M9N2	D2E6A228...	2017-08-31 10:58:47	2017-08-31 10:59:06	3
/Library/Extensions/PromiseSTEX.kext	com.promise.driver.stex	26BCCUR4WN	D2E6A228...	2017-08-31 10:58:47	2017-08-31 10:59:06	3
/Library/Extensions/SoftRAID.kext	com.softraid.driver.SoftRAID	NDGSU3WA4Y	D2E6A228...	2017-08-31 10:58:47	2017-08-31 10:59:05	3

Edit: The SQL database has two tables (kext_load_history_v3 and kext_policy). Let's get going:

1.) open a terminal window and enter:

```
/usr/bin/sqlite3 /var/db/SystemPolicyConfiguration/KextPolicy
```

2.) Enter:

```
sqlite> .tables
```

The result should be: kext_load_history_v3, kext_policy The two tables in the policy database.

We're now going to get some data from the database. Enter one, or both, of the following SQL statements:

```
sqlite> SELECT path, bundle_id, team_id, boot_uuid, created_at, last_seen, flags FROM kext_load_hist
```

Result:

```
/Library/Extensions/CIJUSBLoad.kext|jp.co.canon.ij.print.CIJUSBLoad|XE2XNRRXZ5|D2E6A228-1C85-4138-9C
/Library/Extensions/BJUSBLoad.kext|jp.co.canon.bj.print.BJUSBLoad|XE2XNRRXZ5|D2E6A228-1C85-4138-9CC5
/Library/Extensions/EnergyDriver.kext|com.intel.driver.EnergyDriver|Z3L495V9L4|D4829428-7105-425D-83
/Applications/Intel Power Gadget/EnergyDriver.kext|com.intel.driver.EnergyDriver|Z3L495V9L4|D4829428
```

Or just:

```
SELECT * FROM kext_load_history_v3;
```

Giving you the same output. Let's do that for the policy table as well.

3.) Enter:

```
sqlite> SELECT team_id, bundle_id, allowed, developer_name, flags FROM kext_policy;
```

Result:

```
XE2XNRRXZ5|jp.co.canon.bj.print.BJUSBLoad|1|Canon Inc.|0
XE2XNRRXZ5|jp.co.canon.ij.print.CIJUSBLoad|1|Canon Inc.|0
Z3L495V9L4|com.intel.driver.EnergyDriver|1|Intel Corporation Apps|1
```

Or just:

```
SELECT * FROM kext_policy;
```

Again. Giving you the same kind of output. And with this information we can delete the history of a kext. Let's do that for the Intel EnergyDriver.kext If you do not have this driver installed, then just pick another.

4.) Enter:

```
sqlite> DELETE FROM kext_load_history_v3 WHERE team_id = 'Z3L495V9L4';
```

Done. Let's also delete the policy for this kext.

5.) Enter:

```
sqlite> DELETE FROM kext_policy WHERE team_id = 'Z3L495V9L4';
```

Please note that I used the team_id in my examples, but you can also use the bundle_id or developer_name.

6.) you can now exit sqlite3 by entering:

```
.quit
```

Currently there is (still) no formal way to change the data, but that should change in a future update of High Sierra.

Please note that you can only change the data from the RecoveryOS. Please do not mess with the data. This is just a reference for developers and hackers alike. This is *not* intended for end users!

p.s. I love this one:

"*You can set a firmware password on your Mac to prevent unauthorized changes to NVRAM.*"

Yes. You better do that. Or wait. Just look at that command prompt (Apple-iMacPro:~ Apple) showing you that I changed the setting from a macOS High Sierra terminal window, as administrator 🙂

BUG… BUG… BUGREPORTER !!!

Source: https://pikeralpha.wordpress.com/2017/08/29/user-approved-kernel-extension-loading/