High Sierra's 'Secure Kernel Extension Loading' is Broken

Archived: 2026-04-05 12:40:26 UTC

High Sierra's 'Secure Kernel Extension Loading' is Broken

› a new 'security' feature in macOS 10.13, is trivial to bypass

09/05/2017

love these blog posts? support my tools & writing on patreon! Mahalo :)

Background
With each new release of macOS, Apple introduces new 'built-in' security enhancements...and macOS High Sierra (10.13) is no exception.

In this blog post we'll take a brief look at High Sierra's somewhat controversial "Secure Kernel Extension Loading" (SKEL) feature. Unfortunately while wrapped in good intentions, in it's current implementation, SKEL merely hampers the efforts of the 'good guys' (i.e. 3rd-party macOS developers such as those that design security products). Due to flaws in its implementation, the bad guys (hackers/malware) will likely remain unaffected. While many respected security researchers, system administrators, and macOS developers have voiced this concern, here we'll prove this by demonstrating a 0day vulnerability in SKEL's implementation that decisively bypasses it fully:

$ kextstat

Index Refs Size Wired Name
1 90 0x9e30 0x9e30 com.apple.kpi.bsd
2 8 0x3960 0x3960 com.apple.kpi.dsep

...
130 0 0x4b00 0x4b000 **com.un.approved.kext**

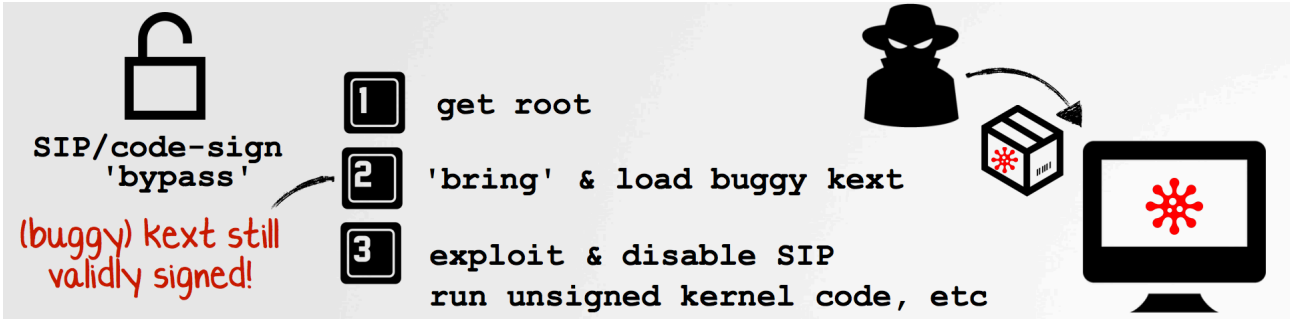
Documented in Apple's Technical Note TN2459, Secure Kernel Extension Loading, is "a new feature that requires user approval before loading new third-party kernel extensions." Other good overviews of SKEL include:

- "Kextpocalypse - High Sierra and Kexts in the Enterprise"
- "Kernel extensions and macOS High Sierra"

While we might initially assume that that the main attack vector SKEL attempts to thwart is the (direct) loading of malicious kernel extensions (i.e. rootkits), I believe this is not the case. First, observe that (AFAIK), we have yet to see any signed kernel-mode macOS malware! Since OS X Yosemite, any kexts have to be signed with a kernel code-signing certificate. And unlike user-mode Developer IDs, Apple is incredibly 'protective' of such kernel code-signing certificates - only giving out a handful to legitimate 3rd-party companies that have justifiable reasons to create kernel code. As security features are often costly to implement, they are generally introduced to reactively address widespread issues. (Unless they are introduced as a control mechanism, under the guise of a 'security feature' (*cough cough*)).

Instead the main (security) goal of SKEL is to block the loading of legitimate but (known) vulnerable kexts. Until Apple blacklists these kexts via the OSKextExcludeList dictionary (in AppleKextExcludeList.kext/Contents/Info.plist), attackers can simply load such kexts, then exploit them to gain arbitrary code execution within the context of the kernel. Note that such blacklisting is often is delayed as it can badly break legitimate functionality until the user has upgraded to a non-blacklisted version of the kext.

About a year ago I discussed this attack vector in my DefCon talk, "I got 99 Problems, but Little Snitch ain't one!" (note: this is a well known attack vector to bypass kernel code-signing requirements on both Windows and macOS):



In my talk, I discussed an exploitable kernel heap-overflow in LittleSnitch's kernel driver, LittleSnitch.kext. While this bug could to be abused to escalate privileges on Macs that had LittleSnitch installed, once the vulnerability was patched the bug didn't die. Instead, as shown on the slide, local privileged attackers could still utilize the vulnerable driver to bypass macOS's kernel code-signing requirements. The steps for this attack are as follows:

1. With root privileges, load a vulnerable copy of the LittleSnitch.kext (versions < 3.61)
   This would be allowed as the vulnerable driver was still validly signed.

2. Exploit the heap-overflow to gain arbitrary code execution within the kernel.
   Here in the kernel, one can bypass SIP, load unsigned kexts, and much more!

While yes, "Secure Kernel Extension Loading" can also block the direct loading of maliciously signed kexts, it seems its main aim is to thwart the loading of known vulnerable drivers for malicious purposes.

Secure Kernel Extension Loading
So what happens when a user (or installer, or malware) tries to a load a signed 3rd-party kernel extension on High Sierra for the first time? Well, it will be blocked by SKEL and the user will be alerted, unless it:

- was "already installed at the time of upgrading to macOS High Sierra."

- is signed with the same Team ID as a previously approved extension.

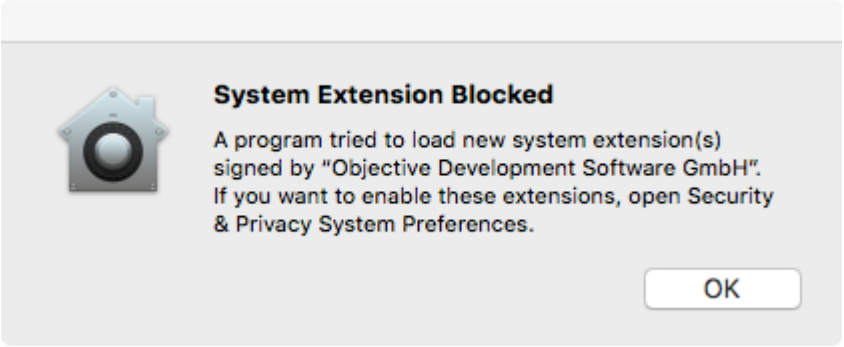
- is "replacing a previously approved extension."
  (I'm guessing based on cryptographically verifiable information such as the Team ID).

- is being loaded on a Mac that is enrolled with an MDM solution.

For example, here we try to manually load the latest version of LittleSnitch.kext on pristine VM instance of High Sierra. Even though the kext is signed with a legitimate (non-revoked) kernel-mode signing certificate and is not blacklisted, SKEL will block it:

kextload LittleSnitch.kext
LittleSnitch.kext failed to load - (libkern/kext) **system policy prevents loading**; check the system/kernel logs for errors or try kextutil(8).

Once the system has blocked the kext from loading, it will also generate an alert to user:



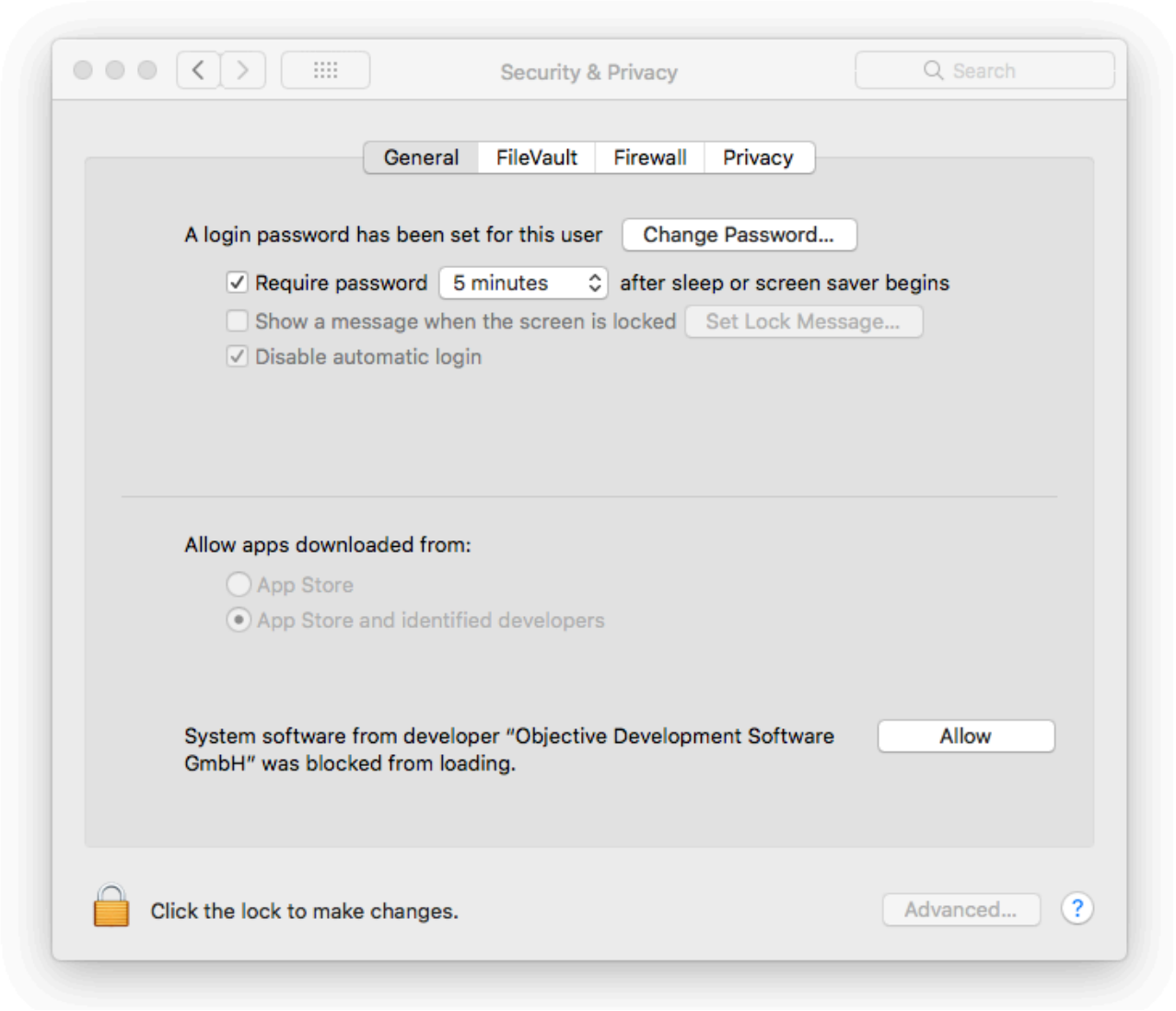
If we monitor file system I/O during this process, we can see both the kext (LittleSnitch.kext) and what appears to be a 'kernel policy' database, are accessed by the system policy daemon, syspolicyd:

```
# fs_usage -w -f filesystem
...
lstat64  /Users/user/Desktop/LittleSnitch.kext  syspolicyd.11844

stat64  /private/var/db/SystemPolicyConfiguration/KextPolicy  syspolicyd.11844
```

If we dump the 'kext policy' database (/private/var/db/SystemPolicyConfiguration/KextPolicy), we can see the LittleSnitch kext has been added to the 'kext_policy' table, but currently it is disallowed ('allowed' is set to 0):

```
# sqlite3 /private/var/db/SystemPolicyConfiguration/KextPolicy '.dump kext_policy'
CREATE TABLE kext_policy ( team_id TEXT, bundle_id TEXT, allowed BOOLEAN, developer_name TEXT,
...);

INSERT INTO kext_policy VALUES('MLZF7K7B5R','at.obdev.nke.LittleSnitch',0,'Objective Development
Software GmbH',4);
```

As was stated in the alert shown to the user, assuming they want the (signed) kernel extension to load, they have to now manually approve it. This is done by launching the System Preferences application (/Applications/System Preferences.app), navigating to the 'Security & Privacy' pane, and in the 'General' tab, clicking 'Allow':



Once the user has allowed the kext to load, the system updates the entry in the 'kext policy' database ('allowed' is set to 1) and allows the kext to load:

sqlite3 /private/var/db/SystemPolicyConfiguration/KextPolicy '.dump kext_policy'
CREATE TABLE kext_policy (team_id TEXT, bundle_id TEXT, **allowed** BOOLEAN, developer_name TEXT, ...);

INSERT INTO kext_policy VALUES('MLZF7K7B5R','at.obdev.nke.LittleSnitch',**1**,'Objective Development Software GmbH',4);

Exploitation

Now, let's crush Secure Kernel Extension Loading 😈 Why? Not because we dislike Apple, but rather to illustrate that in its current implementation the 'bad' guys will have no problem bypassing it. IMHO, this is important for Apple to understand!

Our goal here is to programmatically load a signed kext that has never been installed or loaded on the High Sierra system. This kext could either be a malicious, or more likely a vulnerable one that afford an attacker unfettered kernel access via exploitation. Of course, both of these attacks should be blocked by SKEL. Alas, they are not.

It's important to note that while it is trivial bypass SKEL, Apple did put at least a little effort into thwarting obvious bypasses.

For example, if an attacker could directly modify the 'kext policy' database, obviously they could simply 'pre-approve' kexts. In order prevent this, Apple protects the database with system integrity protection (SIP):

ls -aOl /private/var/db/SystemPolicyConfiguration/KextPolicy
-rw-r--r-- 1 root wheel **restricted** /private/var/db/SystemPolicyConfiguration/KextPolicy

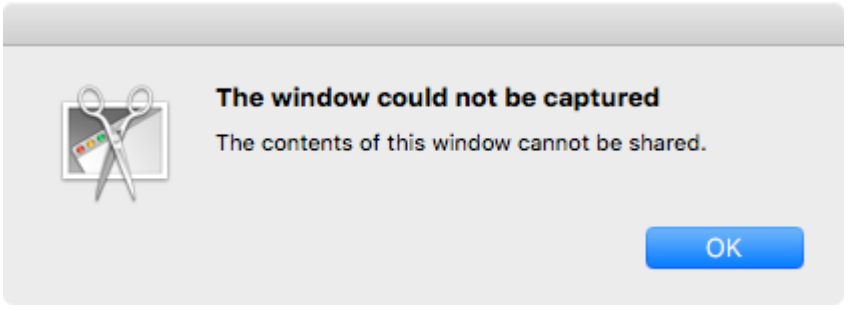
As this database falls under SIP, this means even if an attacker has root privileges they cannot subvert it:

echo "some data" >> /private/var/db/SystemPolicyConfiguration/KextPolicy
sh: /private/var/db/SystemPolicyConfiguration/KextPolicy: **Operation not permitted**

Another obvious attack vector would be to interact with the UI, perhaps programmatically sending a mouse click event to the 'Allow' button in the System Preferences' 'Security & Privacy' pane. However this pane (and other sensitive UI components such as security alerts) are designed to thwart such (simulated/programmatic) interactions in recent versions of macOS. For example, below we can see the OS blocking an apple script which attempts to interact with the 'Security & Privacy' pane:

osascript ~/Desktop/iteractWithUI.scpt>br> iteractWithUI.scpt:467:472: execution error: System Events got an error: **osascript is not allowed assistive access**. (-1719)

Surprisingly (to me), even if one tries to grab a screen capture of the 'Security & Privacy' pane window (via Grab.app, Capture->Window) this fails:

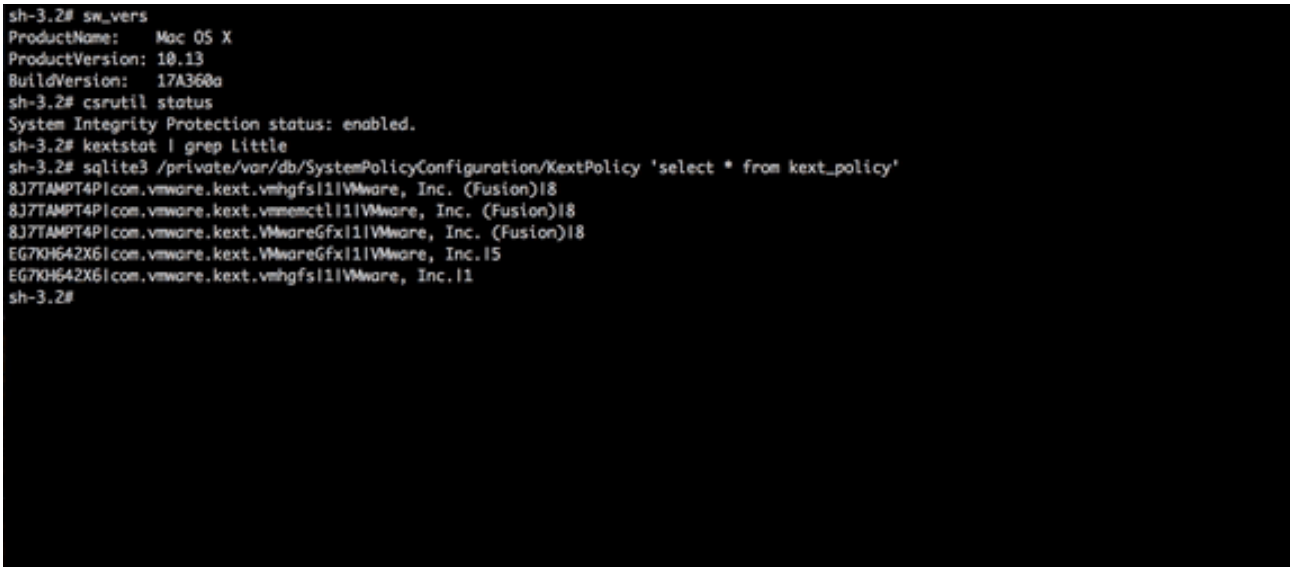


So good news (for Apple), obvious attacks against SKEL fail.

Of course though, as attackers we have the easier job - a single implementation flaw in SKEL may allow us to fully bypass it. Apple on the other hand, has to protect against everything. So, we're always going to win...sometimes after just 20 minutes of poking :P

While at this time I cannot release technical details of the vulnerability, here's a demo of a full SKEL bypass. As can be seen below in the iTerm window below, after dumping the version of the system (High Sierra, beta 9) and showing that SIP is enabled and that kernel extension we aiming to load (LittleSnitch.kext) is not loaded, nor is in

the 'kext policy' database, something magic happens. In short, we exploit an implementation vulnerability in SKEL that allows us to load a new unapproved kext, fully programmatically, without any user interaction:



Conclusion

In this blog post, we briefly discussed High Sierra's "Secure Kernel Extension Loading" (SKEL) and demonstrated a new 0day vulnerability that can be exploited to fully bypass this new 'security' feature.

Unfortunately when such 'security' features are introduced - even if done so with the noblest of intentions - they often just complicate the lives of 3rd-party developers and users without affecting the bad guys (who don't have to play 'by the rules'). High Sierra's SKEL's flawed implementation is a perfect example of this.

Of course if Apple's ultimate goal is simply to continue to wrestle control of the system away from it users, under the guise of 'security', I'm not sure any of this even matters :(

love these blog posts & tools? you can support them via patreon! Mahalo :)
