

# iOS 678 Security

## —== A Study in Fail ==—

Stefan Esser <[stefan.esser@sektion eins.de](mailto:stefan.esser@sektion eins.de)>

# Who am I?

## Stefan Esser

- from Cologne / Germany
- in information security since 1998
- invested in PHP security from 2001 to 20xx
- since 2010 focused on iPhone security (ASLR/jailbreak)
- founder of SektionEins GmbH

# Lots of Fail, but this talk is not about ...

- Apple releasing an iOS 8.0.1 update that would kill cell service on iPhone 6/6p
- Apple adding unsecured double linked lists as meta-data to the iOS 7 kernel heap that made kernel heap exploitation easy (*safe unlink in iOS 8*)
- Apple “improving” their iOS 7 early random number generator so that Tarjei Mandt would totally rip it apart and calculate all past and future numbers
- Apple having a double “goto fail” in their SSL for iOS 6 - iOS 7.0.5 that broke it completely
- **... so yes I am skipping a lot of fail but what I cover is already too much for 45 minutes**

So what is it about?

**this talk is about iOS jailbreaks  
and Apple fixing the bugs within**

# Fixing Bugs ...





BUT APPLE SECURITY  
AIN'T NO JEDI

© New York Jedi/Facebook

So actually ...

**this talk is about iOS jailbreaks  
and Apple **not** fixing the bugs within**

Before we start ...

# **what are the ingredients of an iOS jailbreak?**

## Initial Injection Vector

## Persistency

Not talking about that

## ~~Initial Injection Vector~~

**but apparently Apple had  
to issue several updates to fix those bugs, too**

Concentrating on

Jailbreak

Persistency

# Jailbreak Persistency (I)

- why concentrate on persistency?
- attack surface for injection is just extremely = huge no chance of win
- injection vectors for iOS can be easily exchanged
  - USB, mobile safari, malicious apps
- some injections would require more powerful kernel bugs than public jailbreaks, but those do exist and are traded in the underground

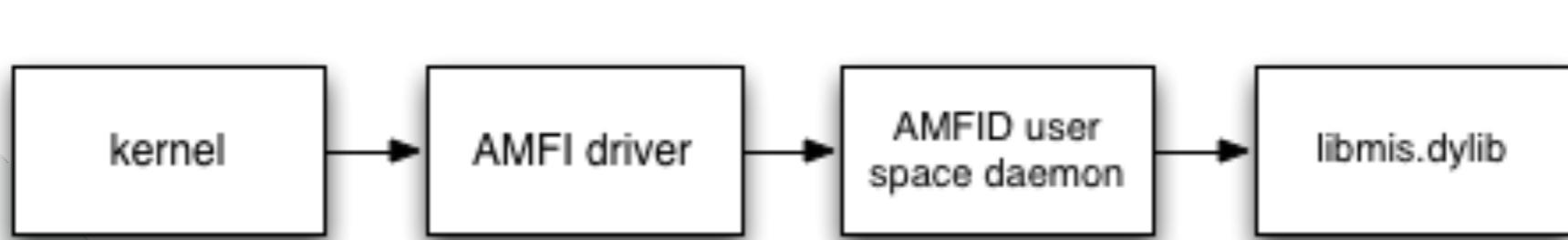
# Jailbreak Persistency (II)

- attack surface for jailbreak persistency is in comparison extremely small
- could be made even smaller and smaller if Apple would do it right
- i just want the public to pressure Apple into changing
- **PUBLIC PERSISTENCE EXPLOITS CAN BE EASILY REUSED IN STATE SPONSORED ATTACKS = LOT CHEAPER + NOT WASTE BUGS**

# AMFI trickery

# AMFI Trickery

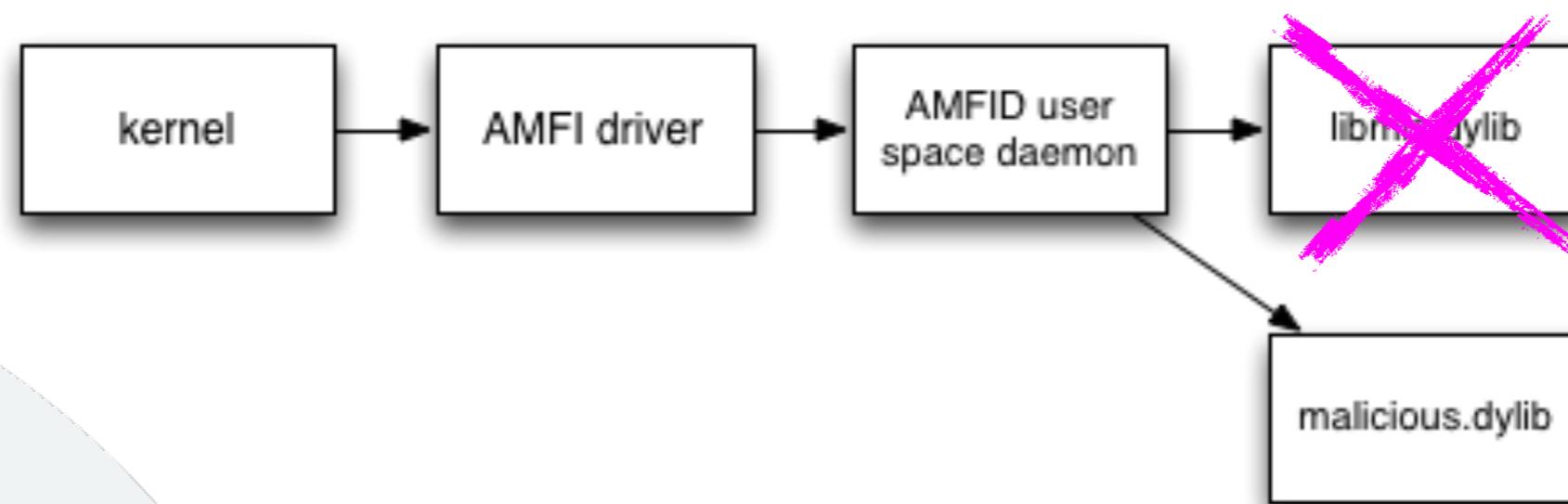
- whole story starts with the exploit chain **evad3rs** “invented” for iOS 6
- for signature verification
  - kernel asks **AMFI** driver
  - **AMFI** driver asks **AMFID** user space daemon
  - **AMFID** uses **libmis.dylib**



- **Attack:**  
inject a malicious library into **AMFID** that makes **libmis.dylib** return “true”

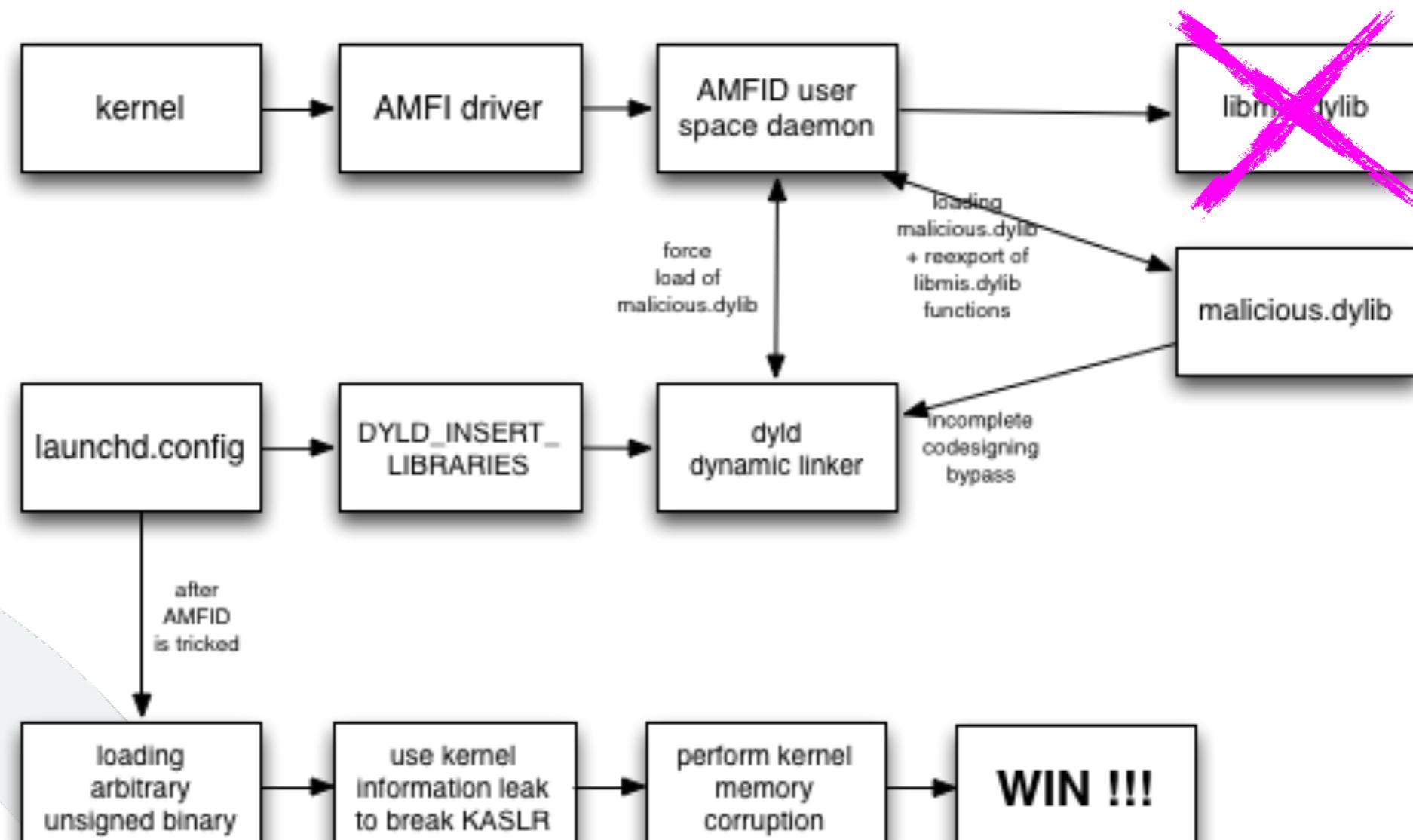
# Injecting a malicious dylib AMFI attack

- injecting malicious **dylib** / replacing **libmis.dylib** not as easy at it sounds
  - need to trick dynamic linker into attempting to do it
  - need to trick dynamic linker into accepting a not signed library



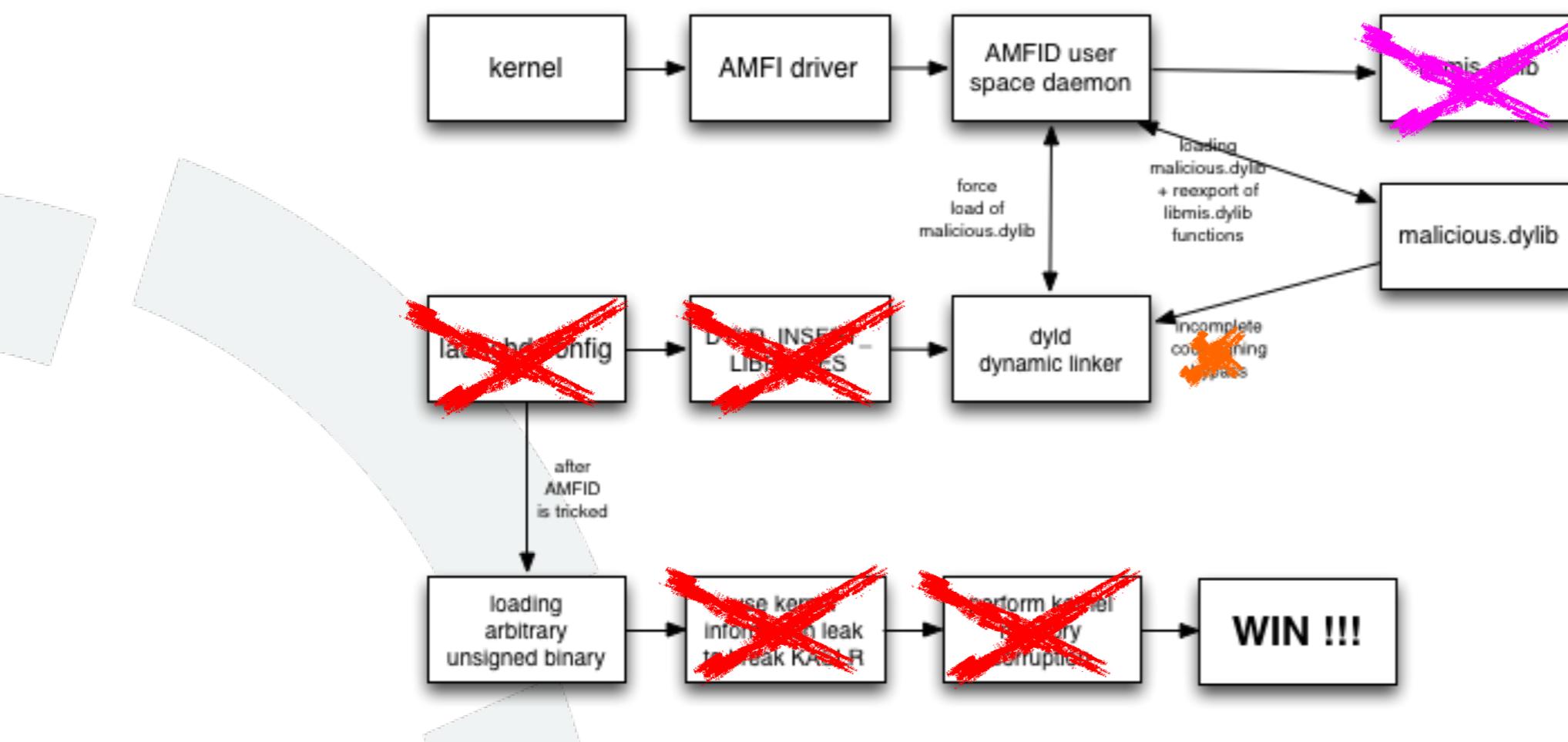
- **Idea:**
  - use **launch daemon config + DYLD\_INSERT\_LIBRARIES** for first part
  - use **incomplete codesigning** bypass for second part

# Putting the full chain together



# Apple “trying” to fix evasi0n

- `launchd.conf` no longer loaded
- \_\_restrict in **AMFID** stops **DYLD\_INSERT\_LIBRARIES**
- apply patch to fix incomplete code signing bypass (patient ALPHA)
- fix kernel info leak and memory corruption



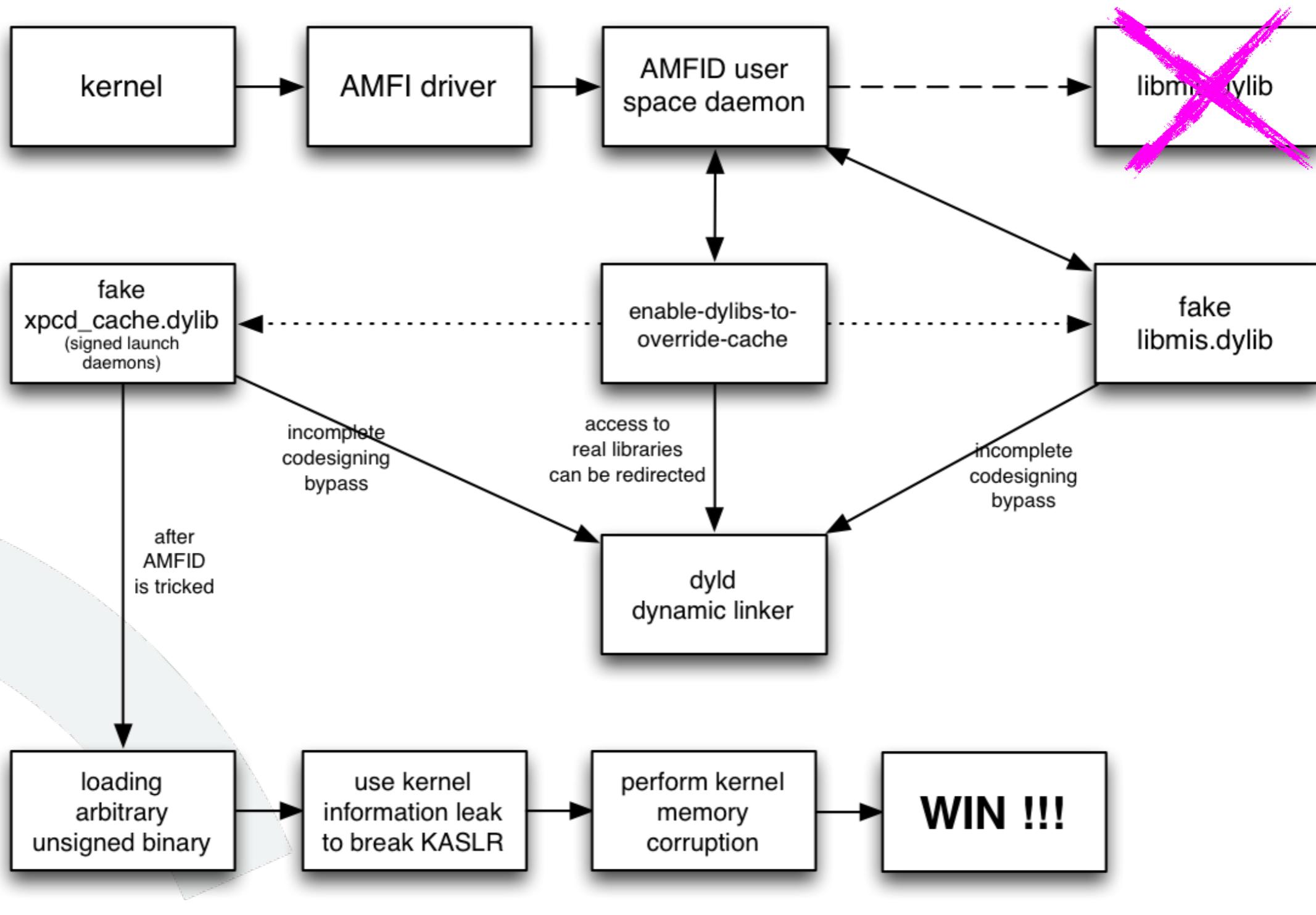
# evad3rs needed new tricks

- patch of incomplete code signing trick (**patient ALPHA**) required new incomplete code signing trick
  - ➡ **evad3rs** unnecessarily burned a new vulnerability some gave to them
- **restrict** in **AMFID** required new way to replace/inject library
  - ➡ just use dyld feature **enable-dylibs-to-override-cache**  
(makes dyld load a *replacement library instead of original one*)

# evad3rs needed new tricks (II)

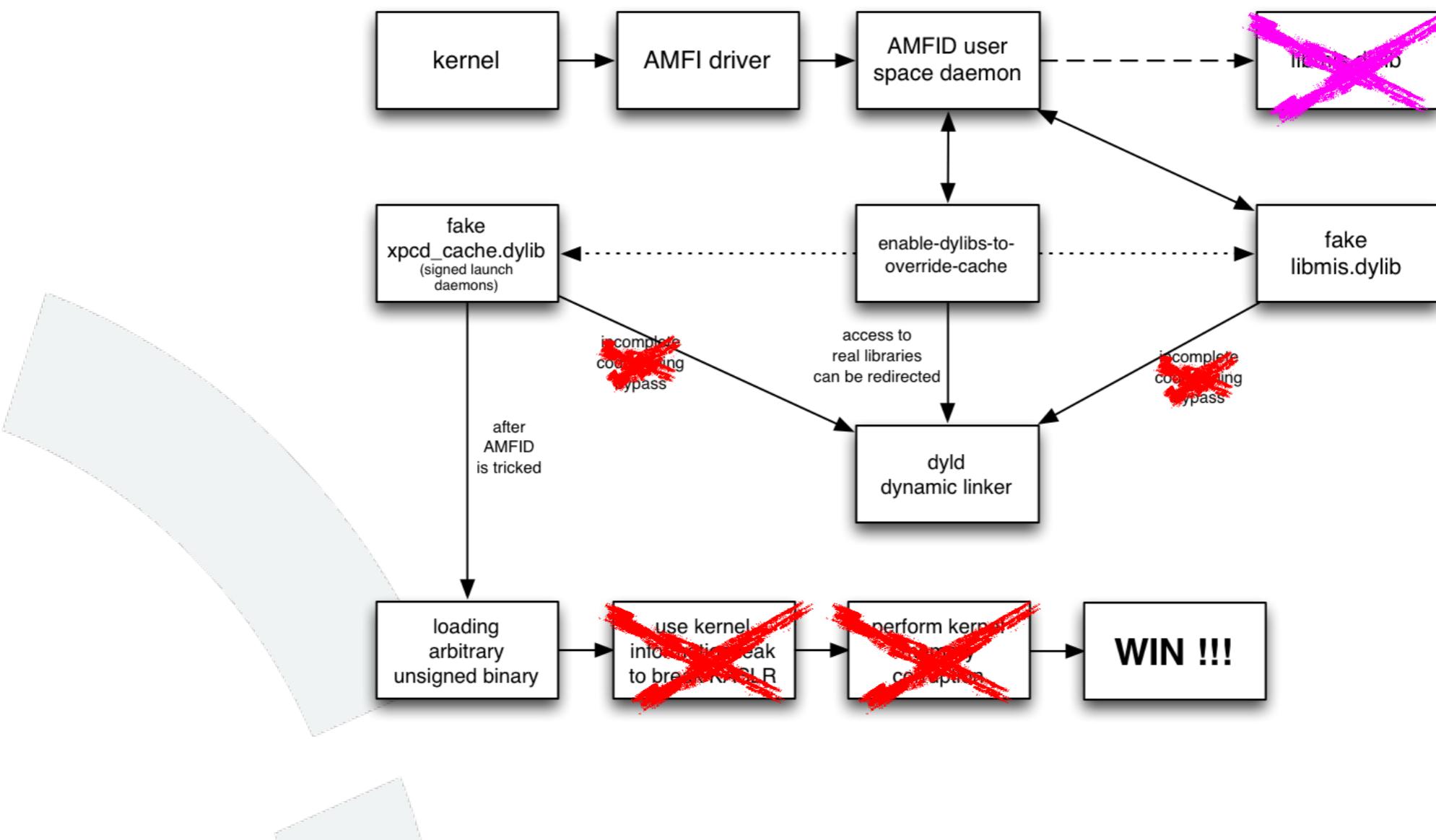
- removal of **launchd.conf** required trick to inject launch daemons beside launch daemon codesigning
  - just use previous two also load a malicious **xpcd\_cache.dylib**
- kernel bugs required to be replaced
  - they just used some new bugs (because there are plenty)

# New chain for evasi0n 7



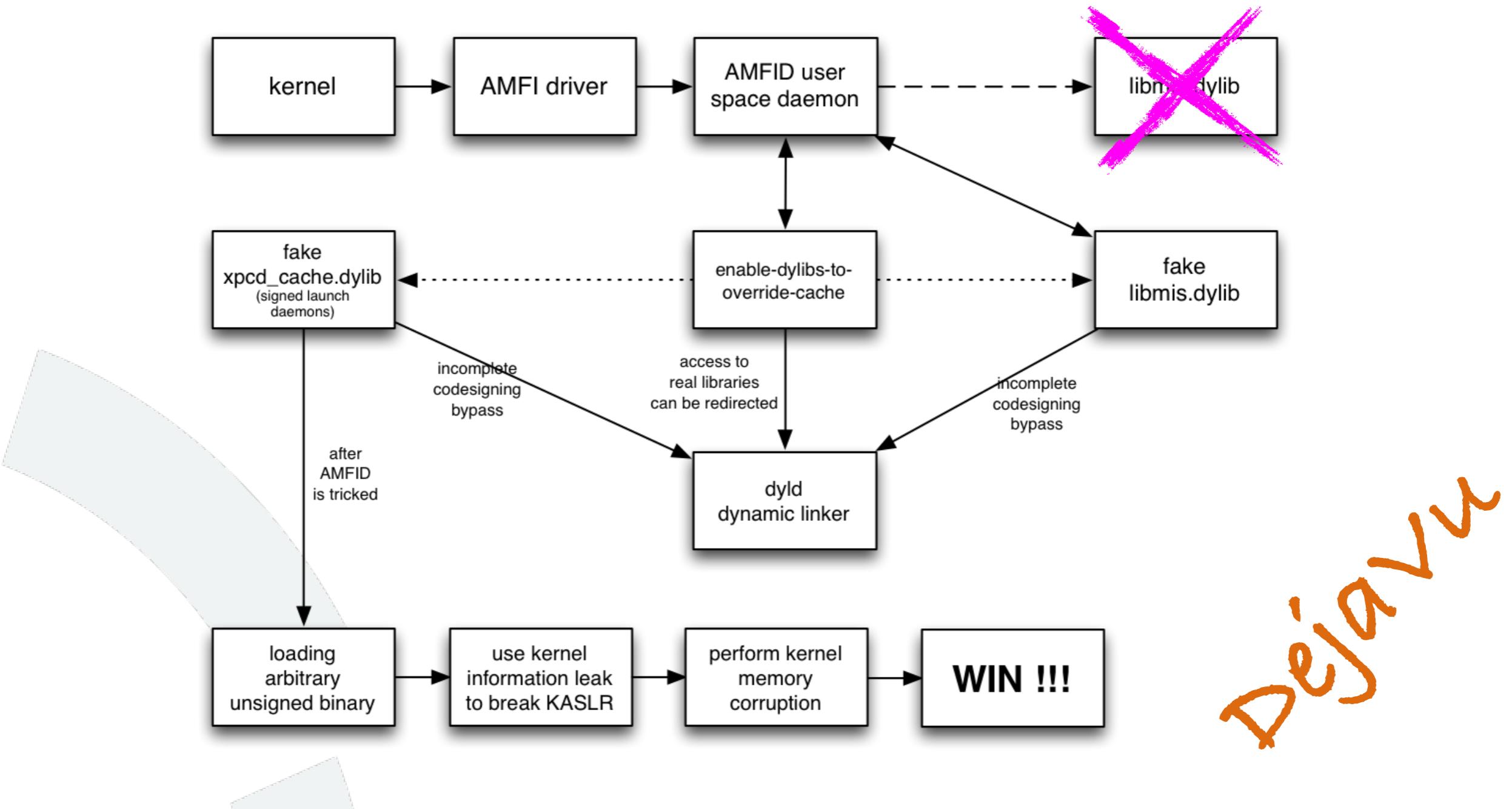
# Apple fixing evasi0n7 \*LESS AGGRESSIVE\*

- new incomplete codesigning bypass trick fixed
- kernel bugs are fixed
- rest of chain is left completely untouched



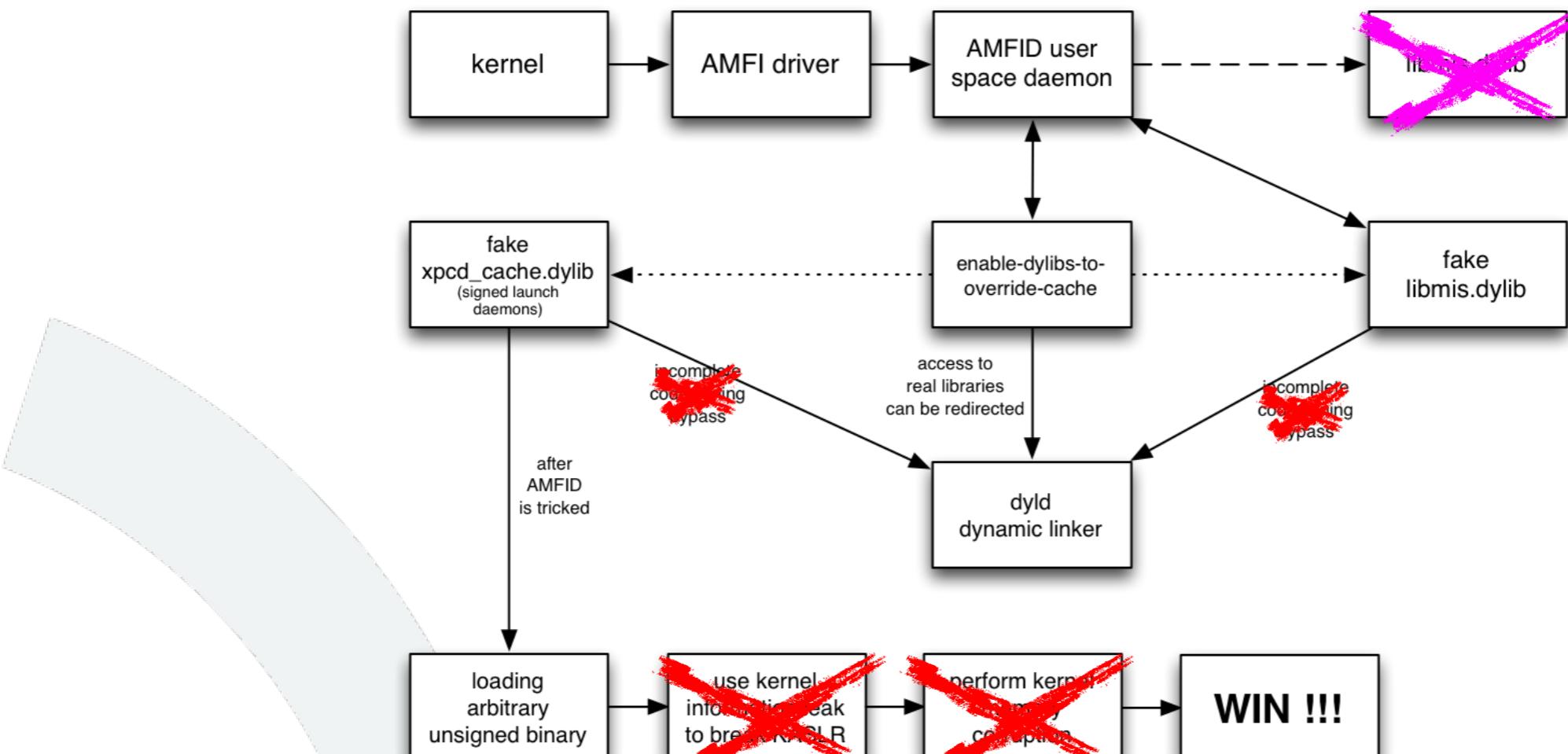
# Pangu7 Jailbreak

- Pangu realised that original bug **Patient ALPHA** was fixed incorrectly
- used new kernel bugs



# Apple “trying to fix” Pangu7

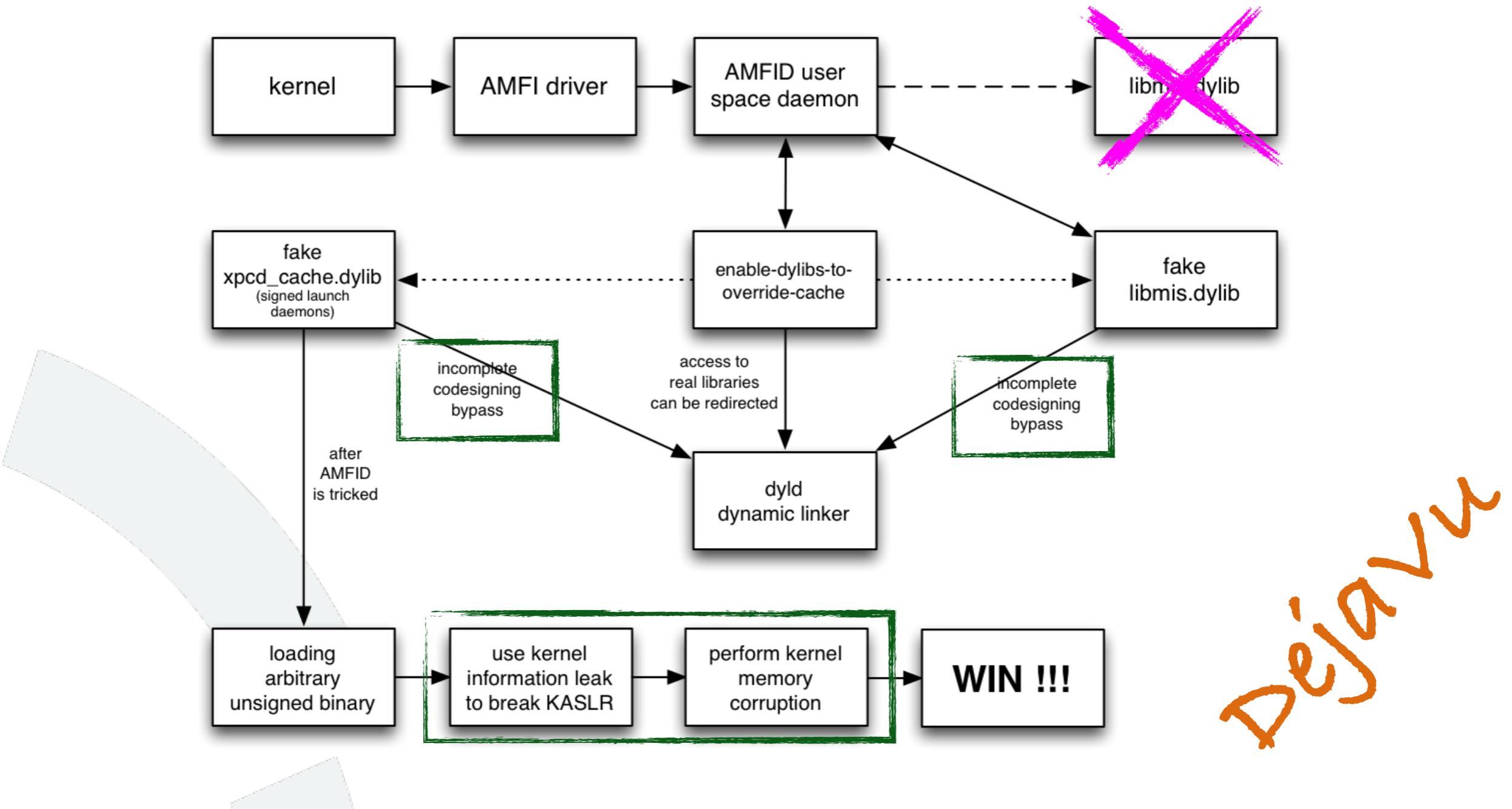
- they try to fix their broken fix for the codesigning
- kernel bugs are claimed fixed
- **rest of chain is still completely untouched**



DejaVu

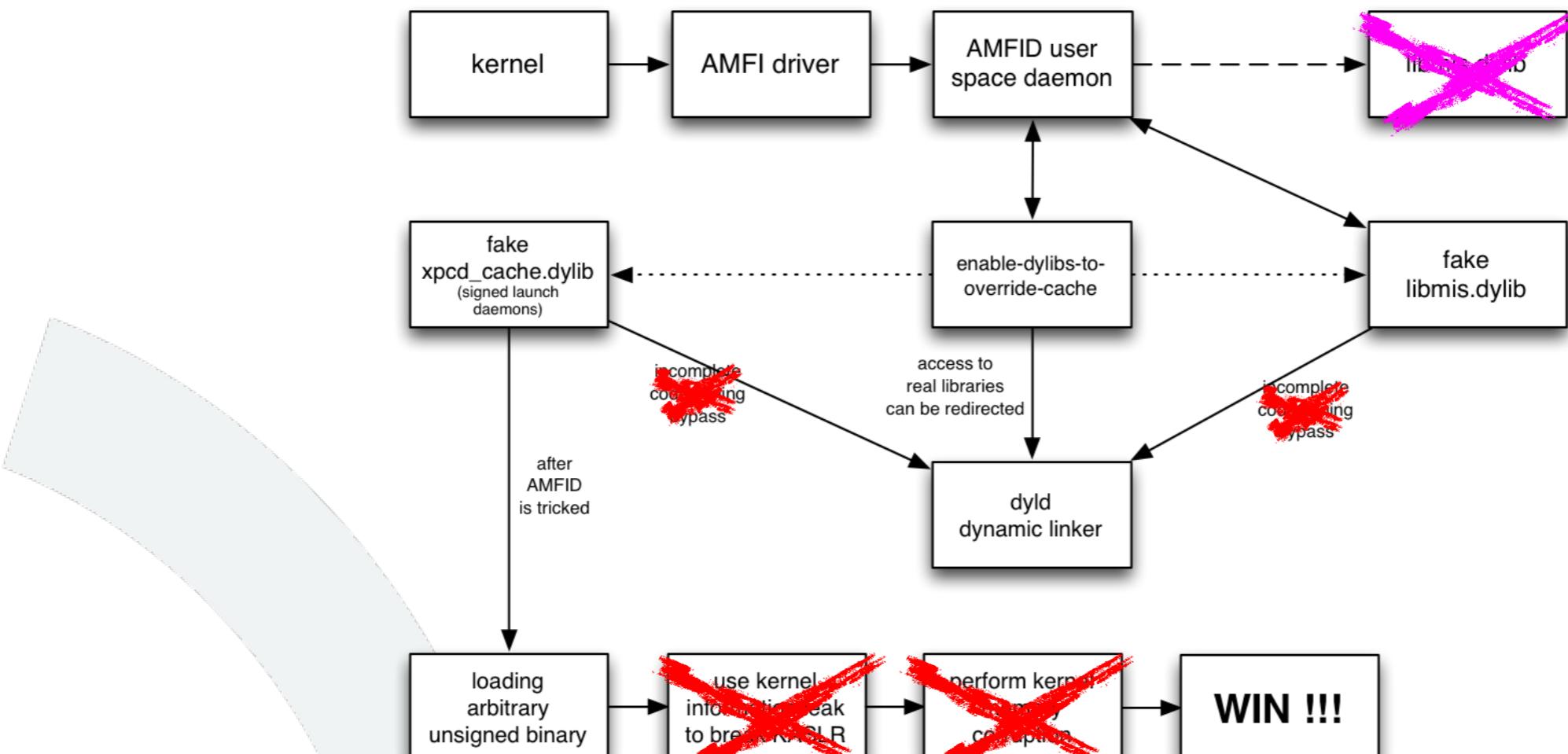
# Pangu8 Jailbreak

- Pangu realised that fix for fix for **Patient ALPHA** is still incorrect
- also their kernel bugs were apparently not patched correctly



# Apple “trying to fix” Pangu8

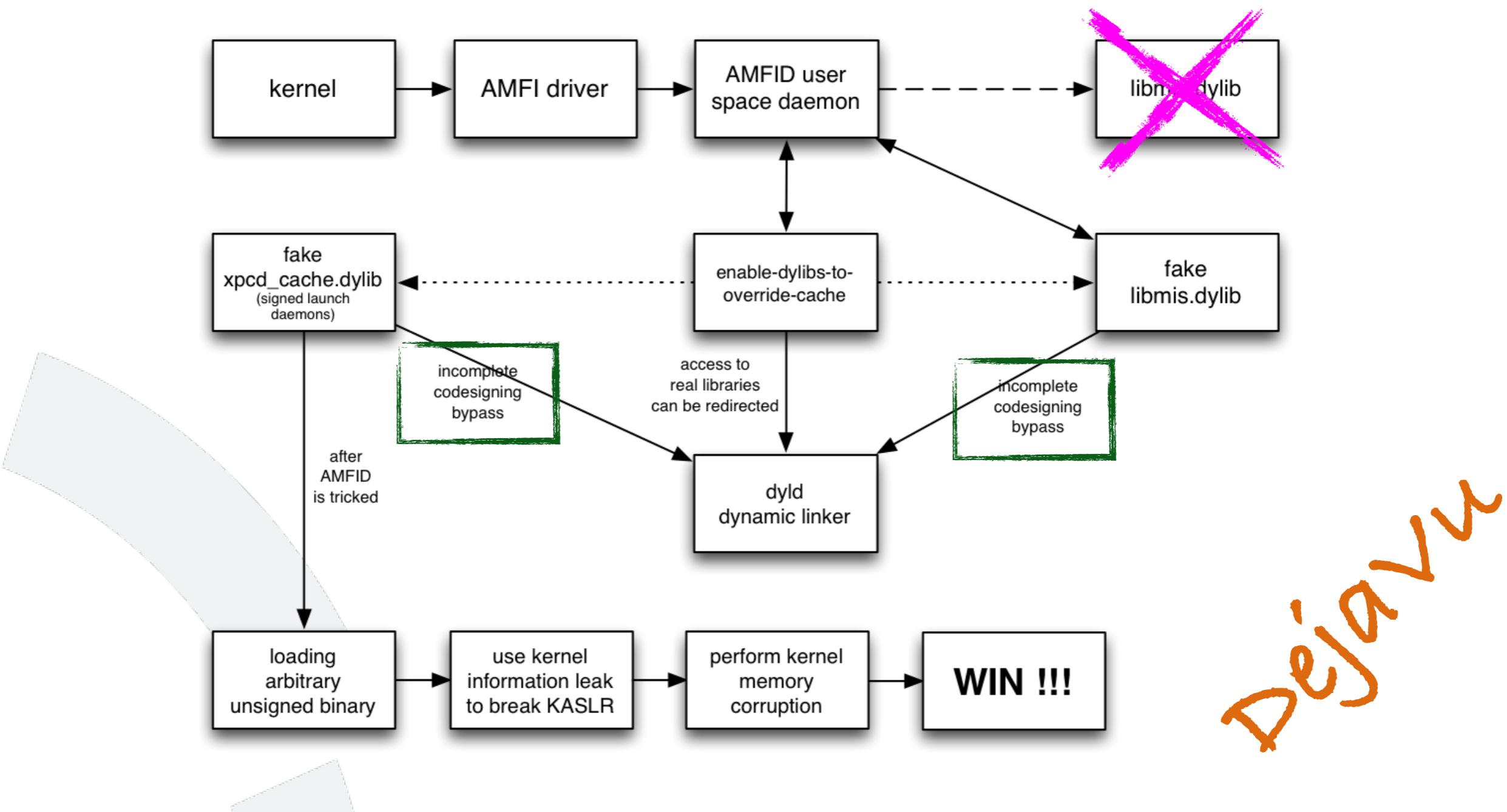
- they try to fix their broken fix for the broken fix for the codesigning
- kernel bugs are claimed fixed again
- **rest of chain is still completely untouched**



DejaVu

# TaiG Jailbreak

- Pangu TaiG realised that fix for fix for fix for fix for **Patient ALPHA** is still incorrect
- used other kernel bugs



# **enable-dylibs-to-override-cache**

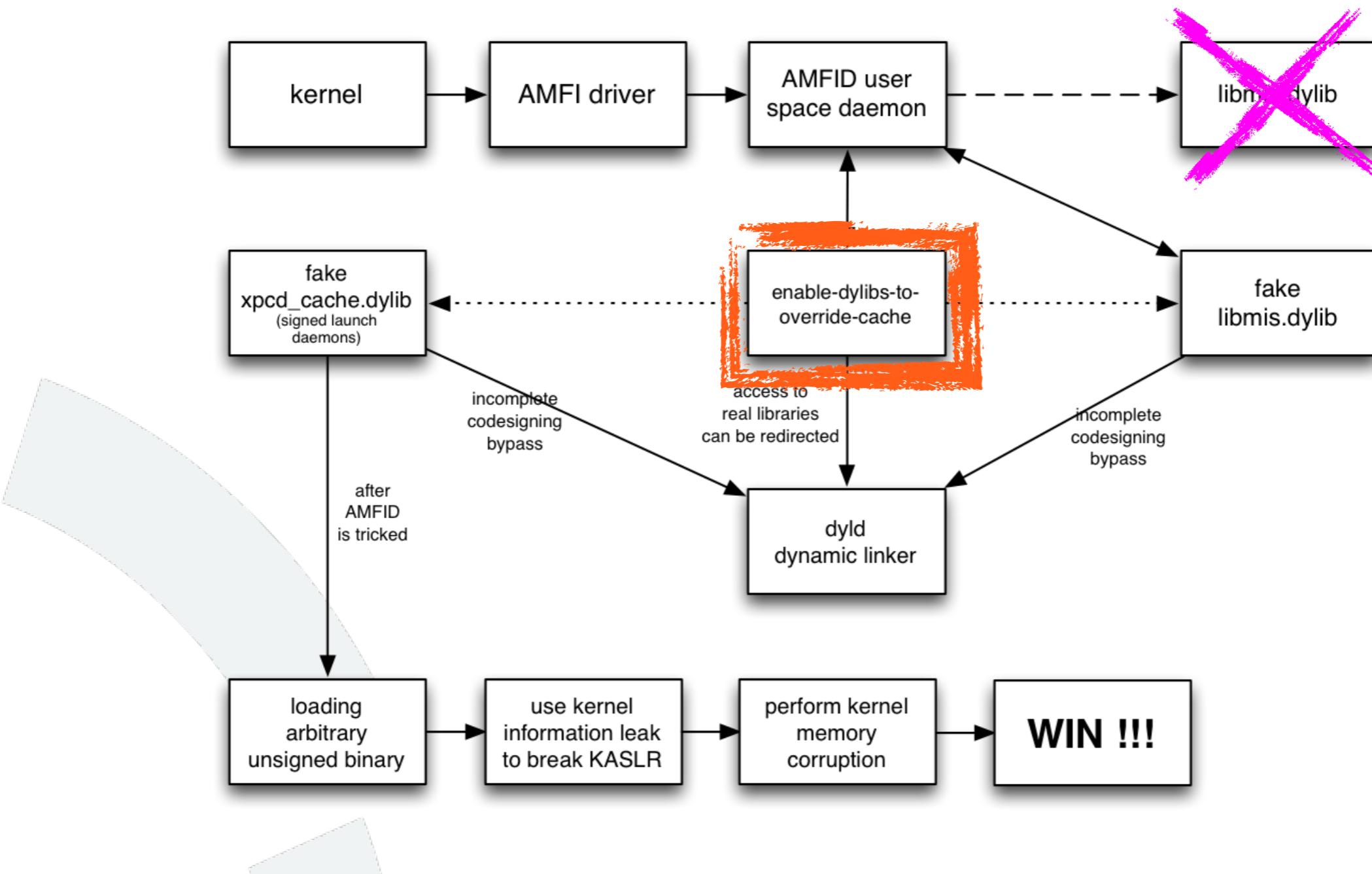


# enable-dylibs-to-override-cache

- a flag file in **/System/Library/Caches/com.apple.dyld**
- when present libraries on disk will be loaded instead of a cached copy
- used by jailbreaks to trick dyld into loading
  - malicious **libmis.dylib** into AMFID
  - malicious list of launchd daemons **xpcd\_cache.dylib** into **launchctl**

# enable-dylibs-to-override-cache

- see how **enable-dylibs-to-override-cache** is in center of everything?
- without this element the whole chain would have been destroyed



# enable-dylibs-to-override-cache

- first abused in **iOS 7.0 jailbreak by evad3rs (December 2013)**
- without it the whole exploit chains of all jailbreaks since then would have fallen apart
- after **14 months still unfixed**
- from a strategic standpoint it is unbelievable that Apple did not fix this
  - by not fixing this for 14 months Apple made persistence on device easier
  - fixing this bug would have made iOS far more secure than the security by obscurity patches that Apple implemented to stop the root filesystem from being writable or the team identifier hack

# Patient Alpha

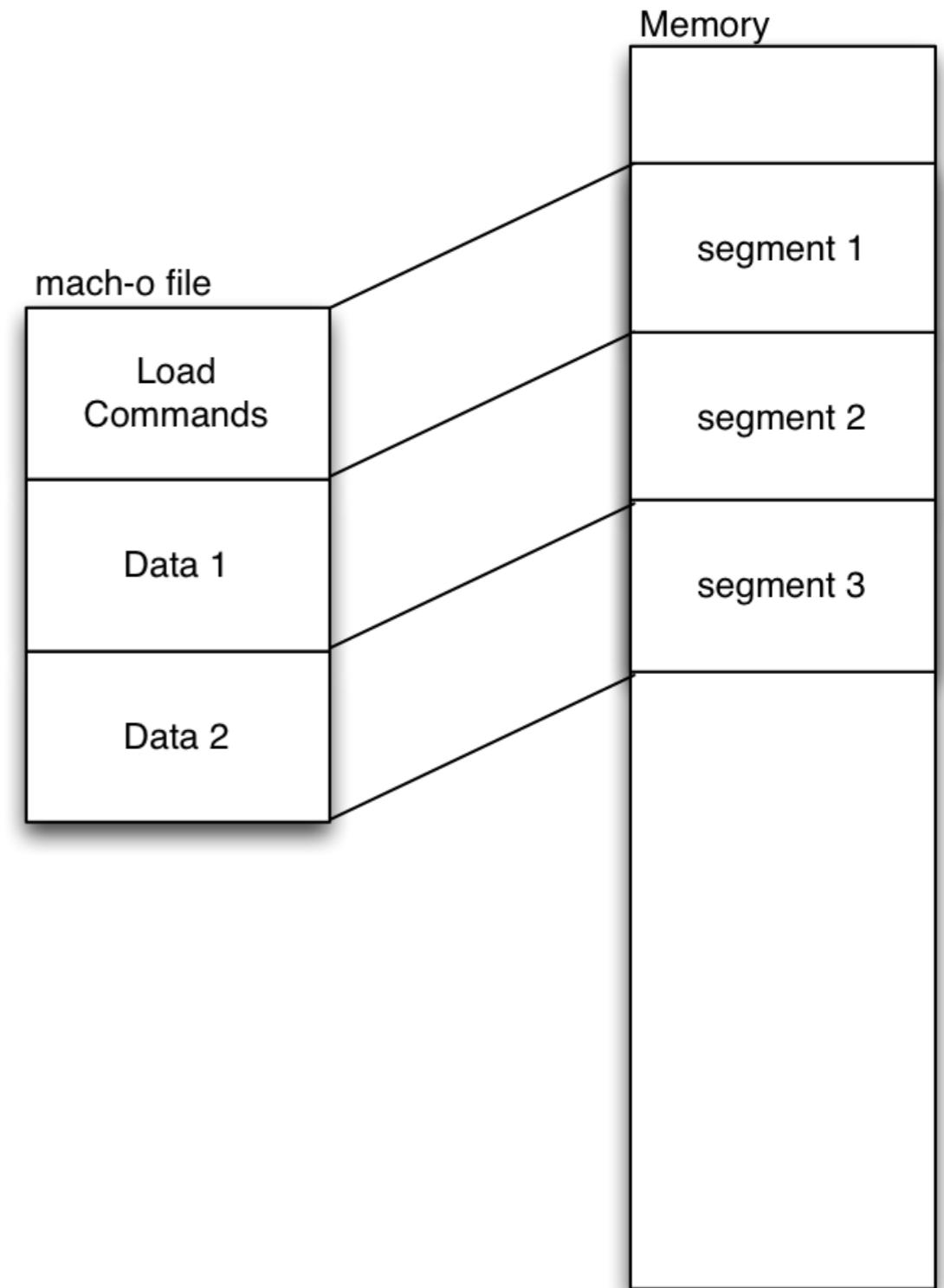
(incomplete codesigning)

# Incomplete Codesigning

- **simplified:**
  - if a page is not executable a missing code sig is not a problem
  - if a page is executable there must be a code sig on first access
- prior to **iOS 5** therefore jailbreaks would use **ALL DATA dylibs** to exploit dyld via various meta-data structures
- around the **end of iOS 4** Apple added checks to dyld to **enforce** load commands are in an **executable segment**
- therefore while header parsing (first access) code sig is required

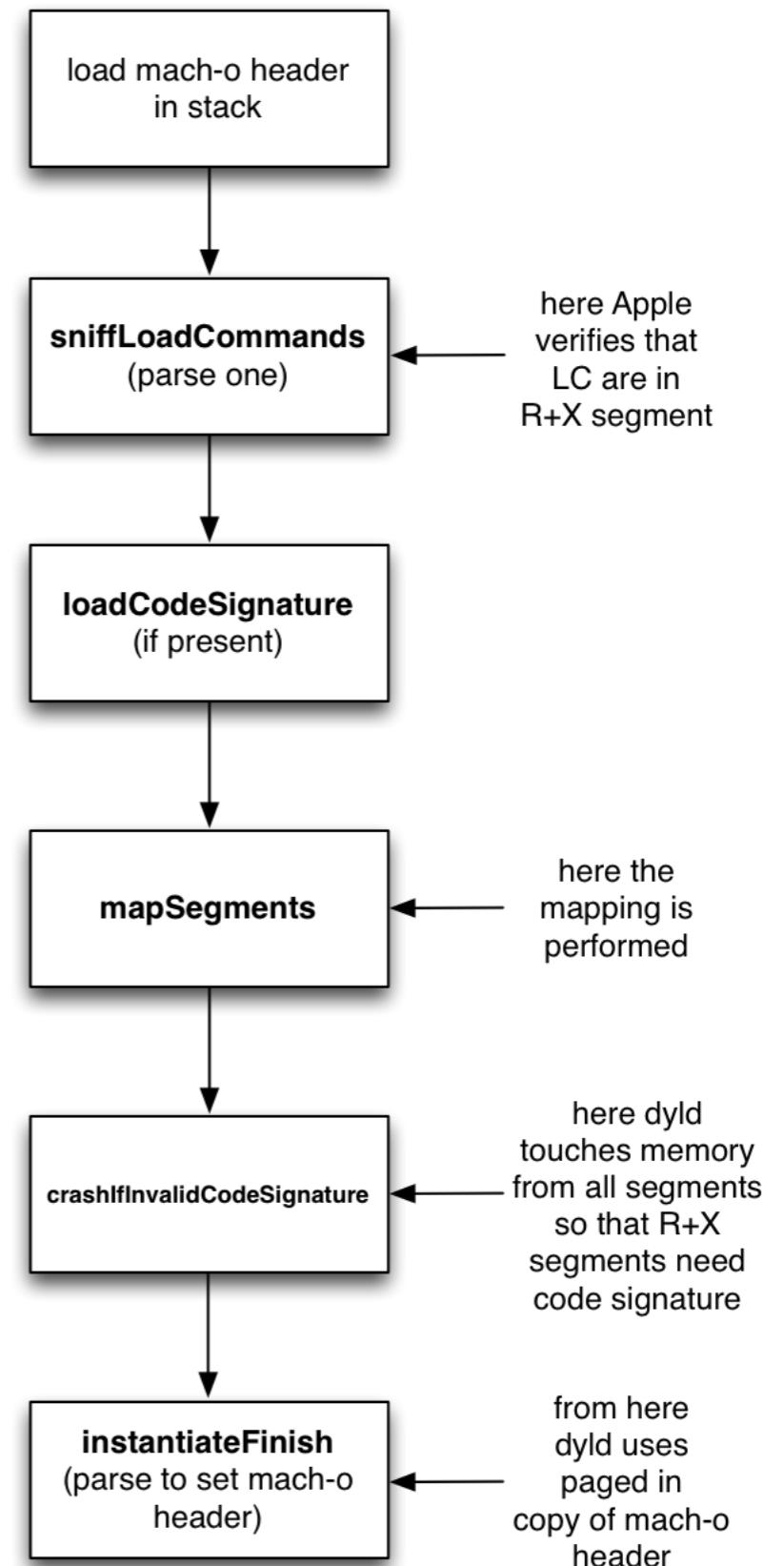
# mach-o dynamic library loading

- **mach-o** dynamic libraries loaded by **dyld**
- **load commands** describe i.a. layout of segments in memory
  - **virtual address** and **virtual size** of segments
  - **file position** and **file size** of segment



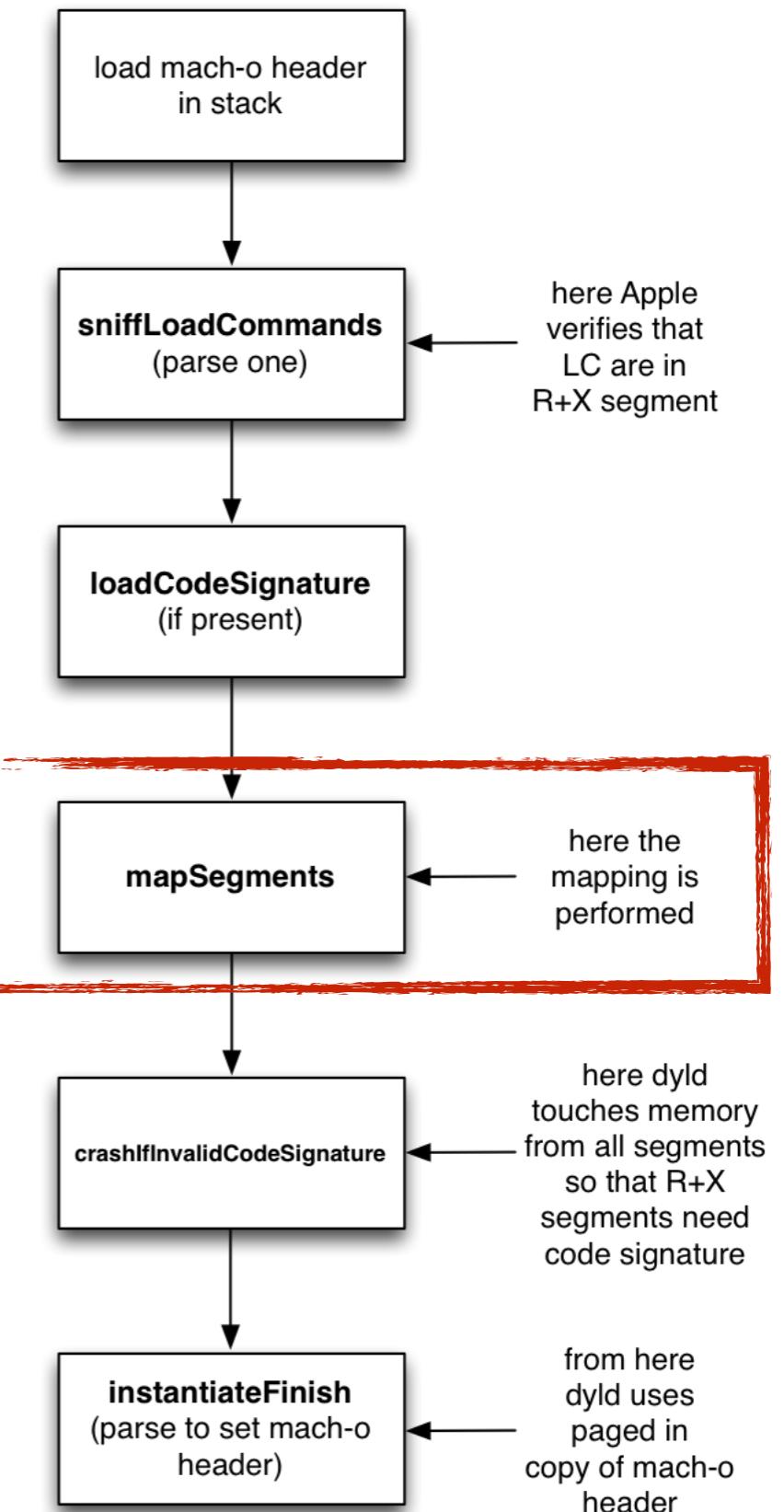
# Ohhh really?

- actually it is not that simple
- mach-o header is first loaded into stack
- initial LC parse is performed to collect info
- this info is used to map the file into memory
- segments are touched to enforce code sig
- another LC parse is performed to make dyld use the mach-o header from paged memory
- more and more parsing



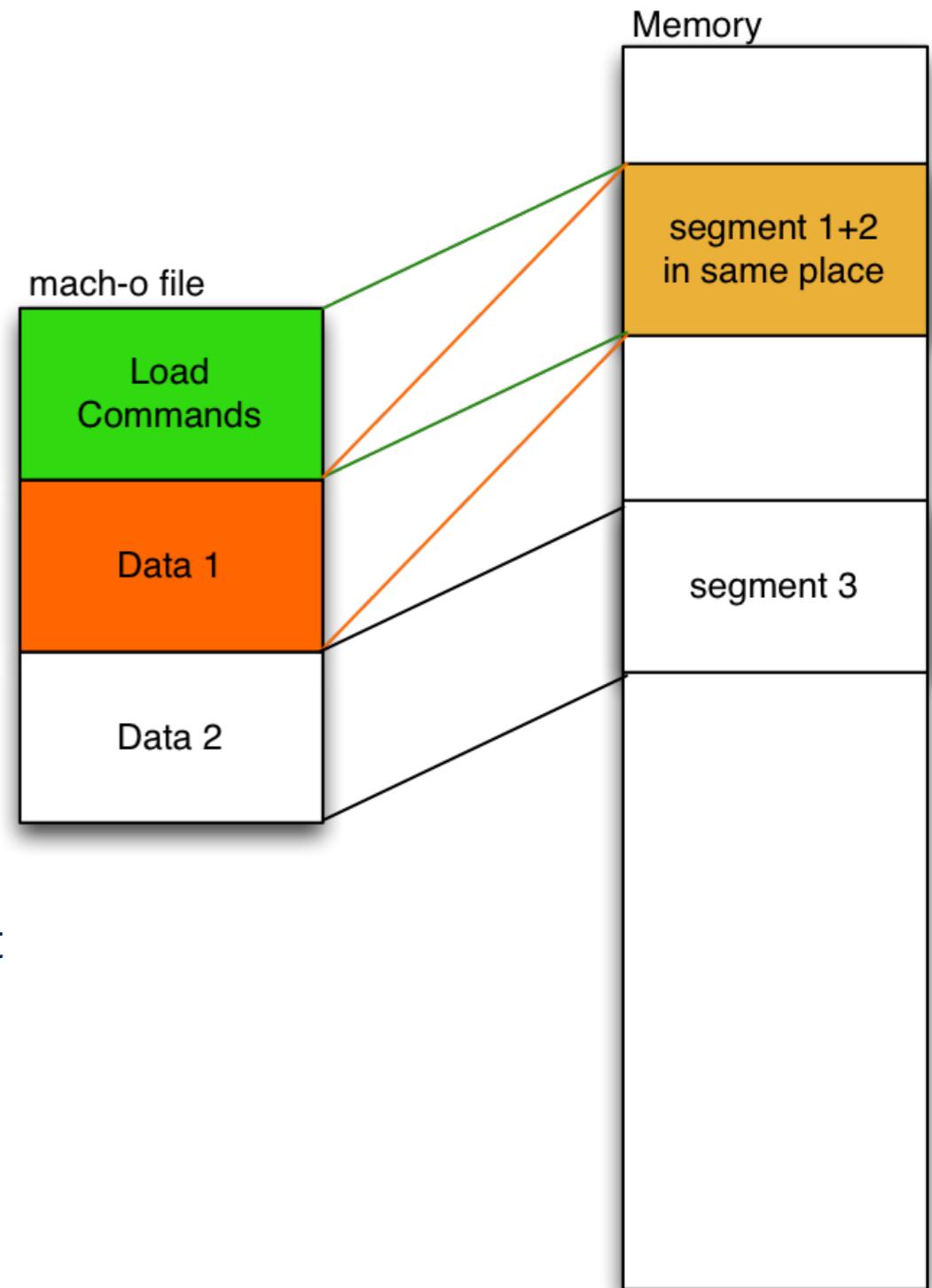
# evad3rs come along

- and then the **evad3rs** came along
- for **evasi0n (iOS 6)** they used a **TOCTOU** trick to bypass the checks
- attack logic between **sniffLoadCommands** and **crashIfInvalidCodeSignature**
- just trick **mapSegments**



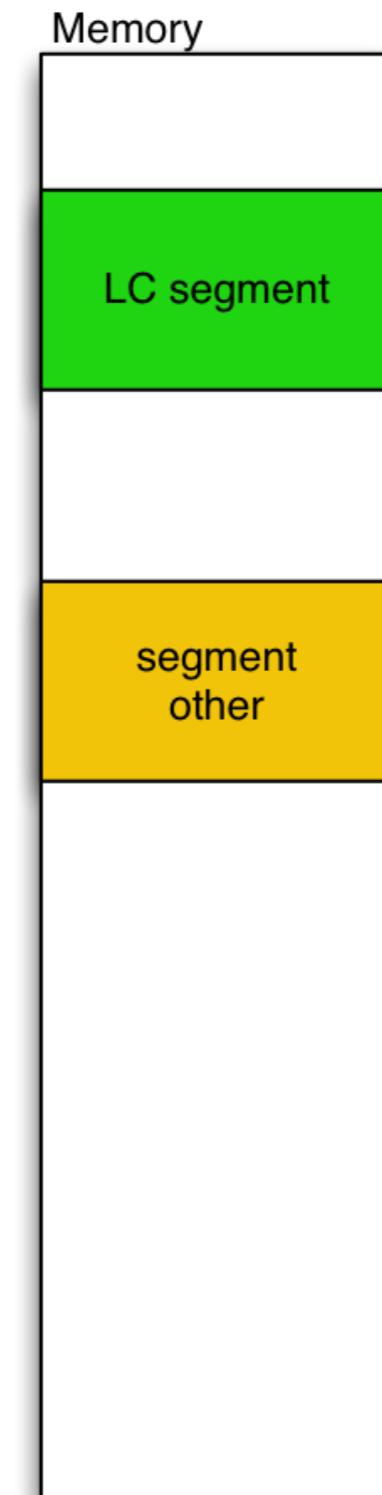
# mapSegments

- **mapSegments** uses **mmap()**
- **mmap** allows to override previous mappings
- just have two mappings at same virtual address
  - 1st contains LC and is **R+X**
  - 2nd contains fake LC and is **R / RW**
- therefore at time of later access to page it will not require a code signature



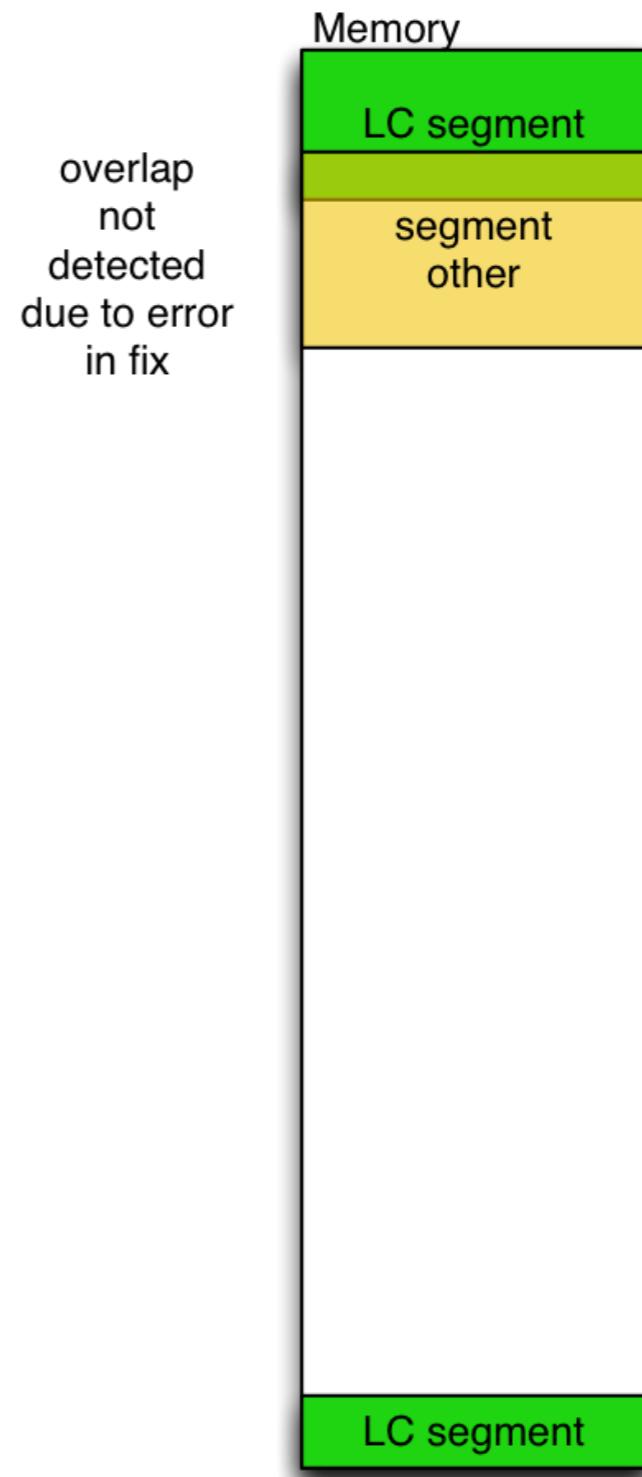
# Apple's first fix

- Apple added a detection for overlapping segments
- code tests **LC segment** against all others so that it does not get overlapped by other segment
- test uses **virtual address** and **virtual size**



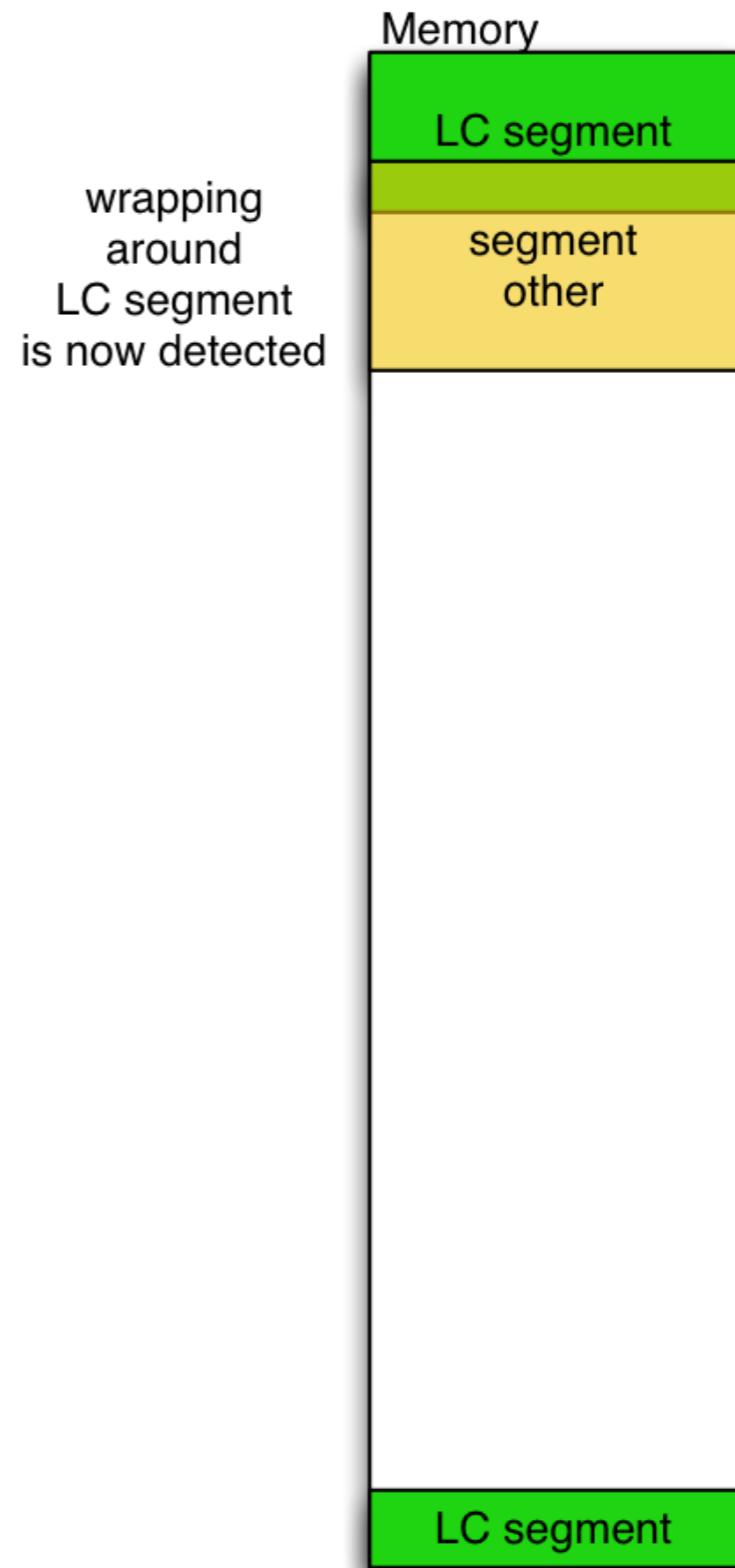
# Problem with Apple's first fix

- the security check by Apple did not account for integer wraps
- a segment with **vmaddr** at end of address space and wrapping around is not detected as overlapping
- in reality the mapping is not wrapping around the address space due to being slid for **ASLR**



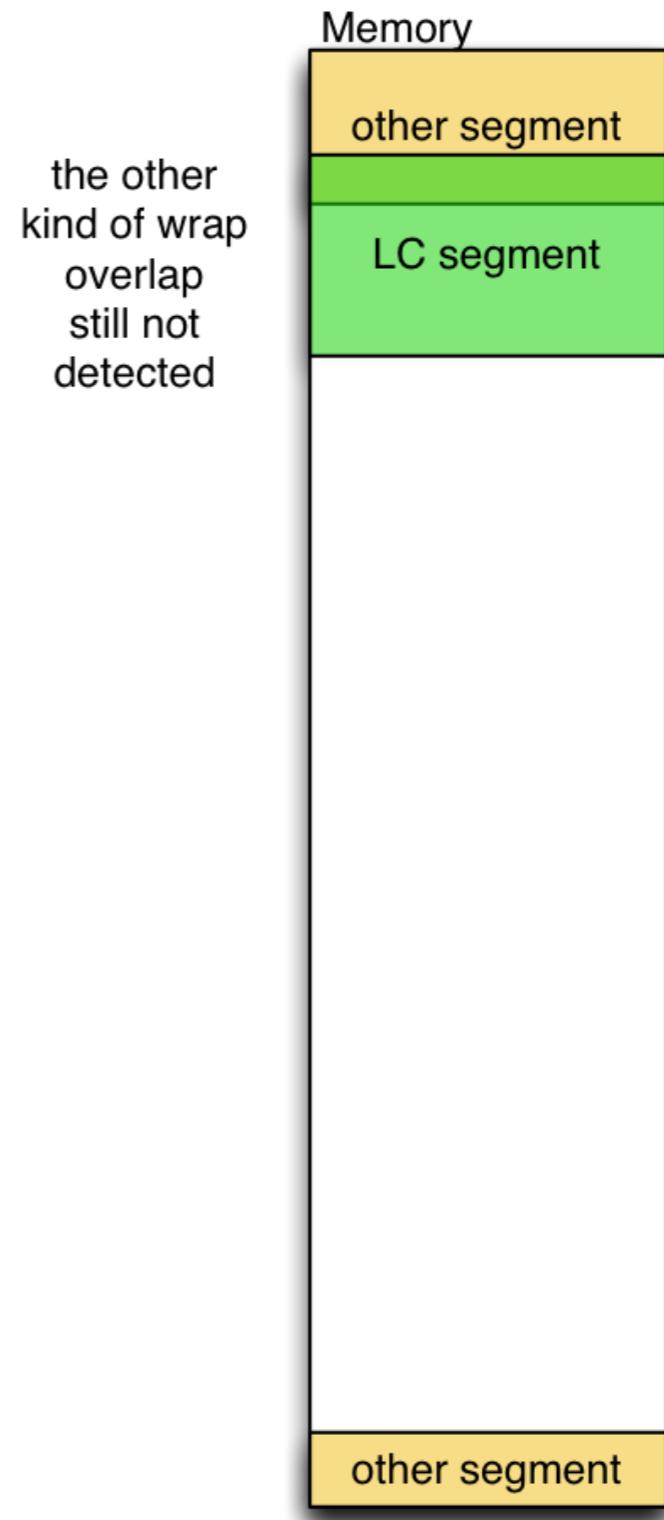
# Apple's second fix

- Apple added a detection for LC segment wrapping around
- such a setup will be detected and disallowed



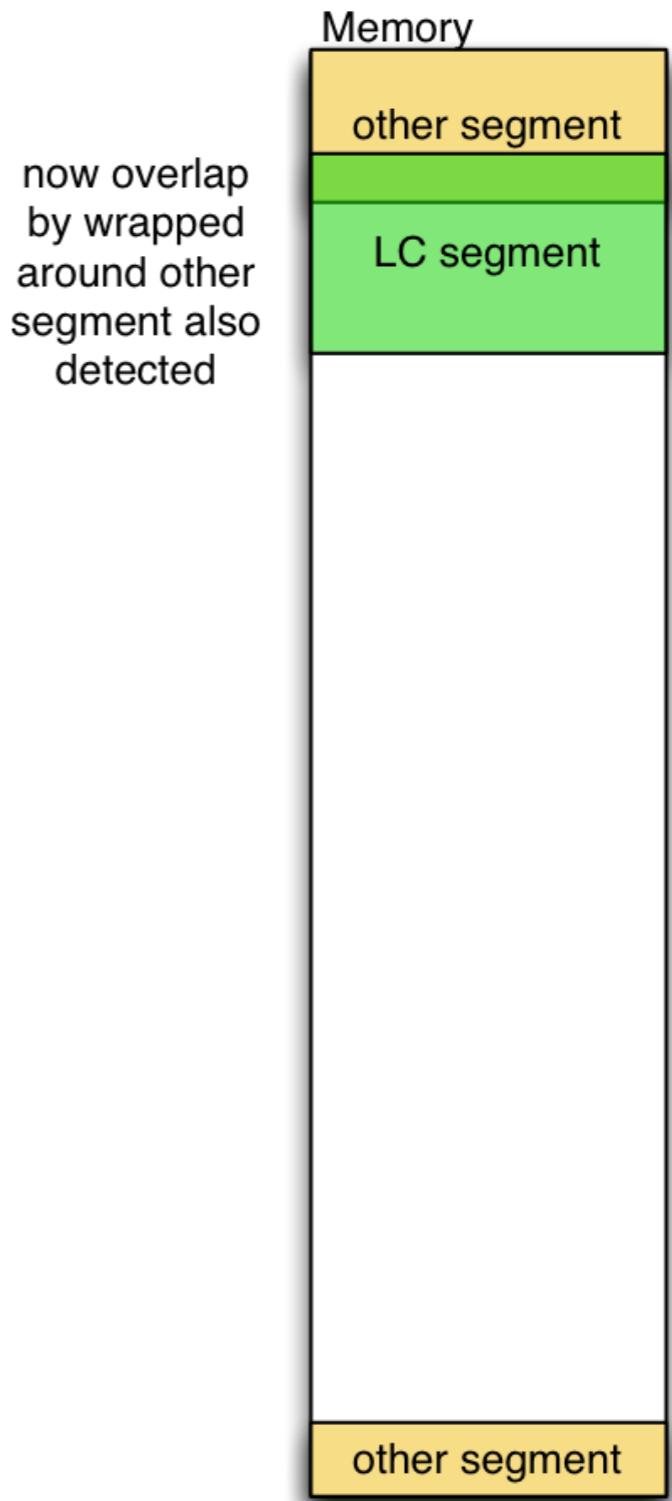
# Problem with Apple's second fix

- the integer wrap check was only done on the LC segment
- just doing the same trick again but letting the other segment wrap around would still do the trick
- what was Apple thinking?



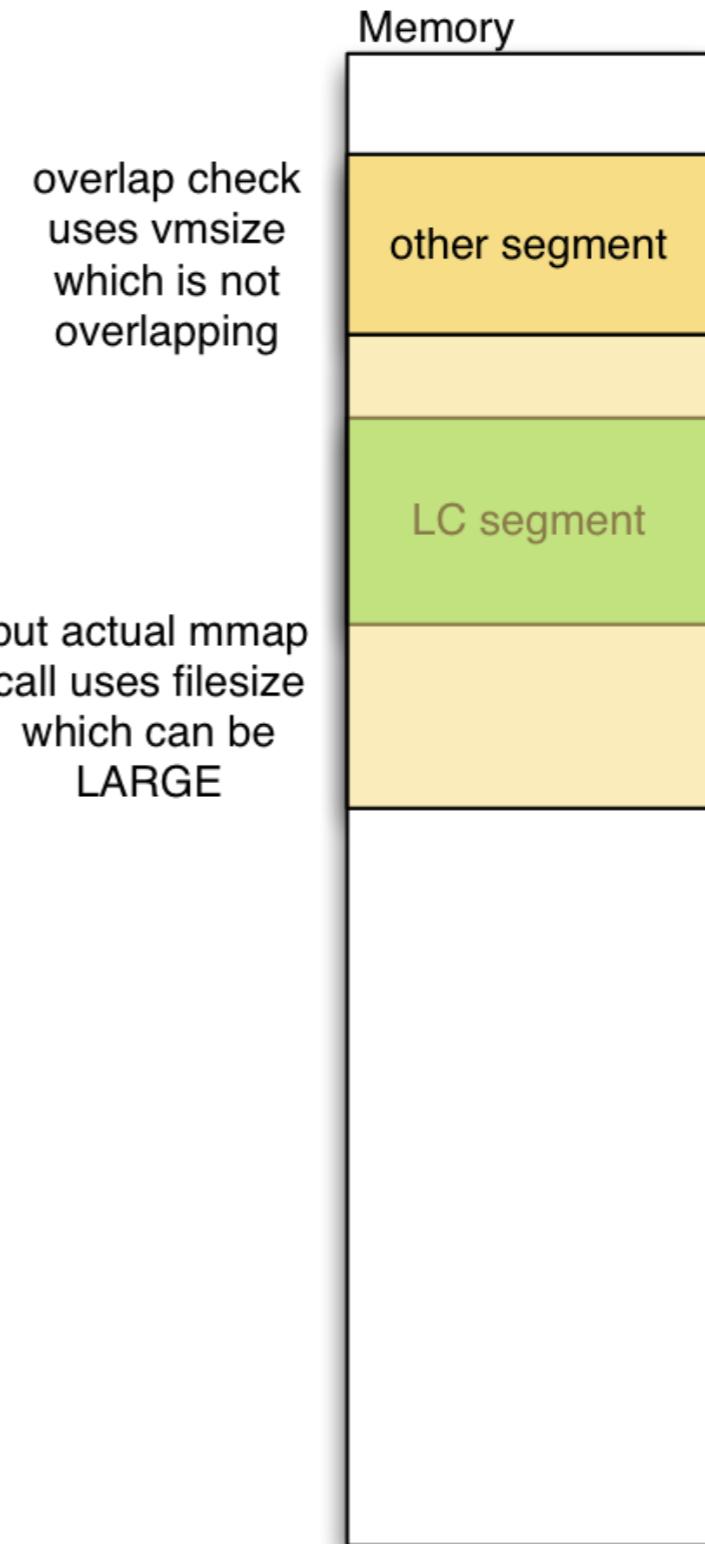
# Apple's third fix

- Apple added a detection for other segments wrapping around
- such a setup will now also be detected and disallowed



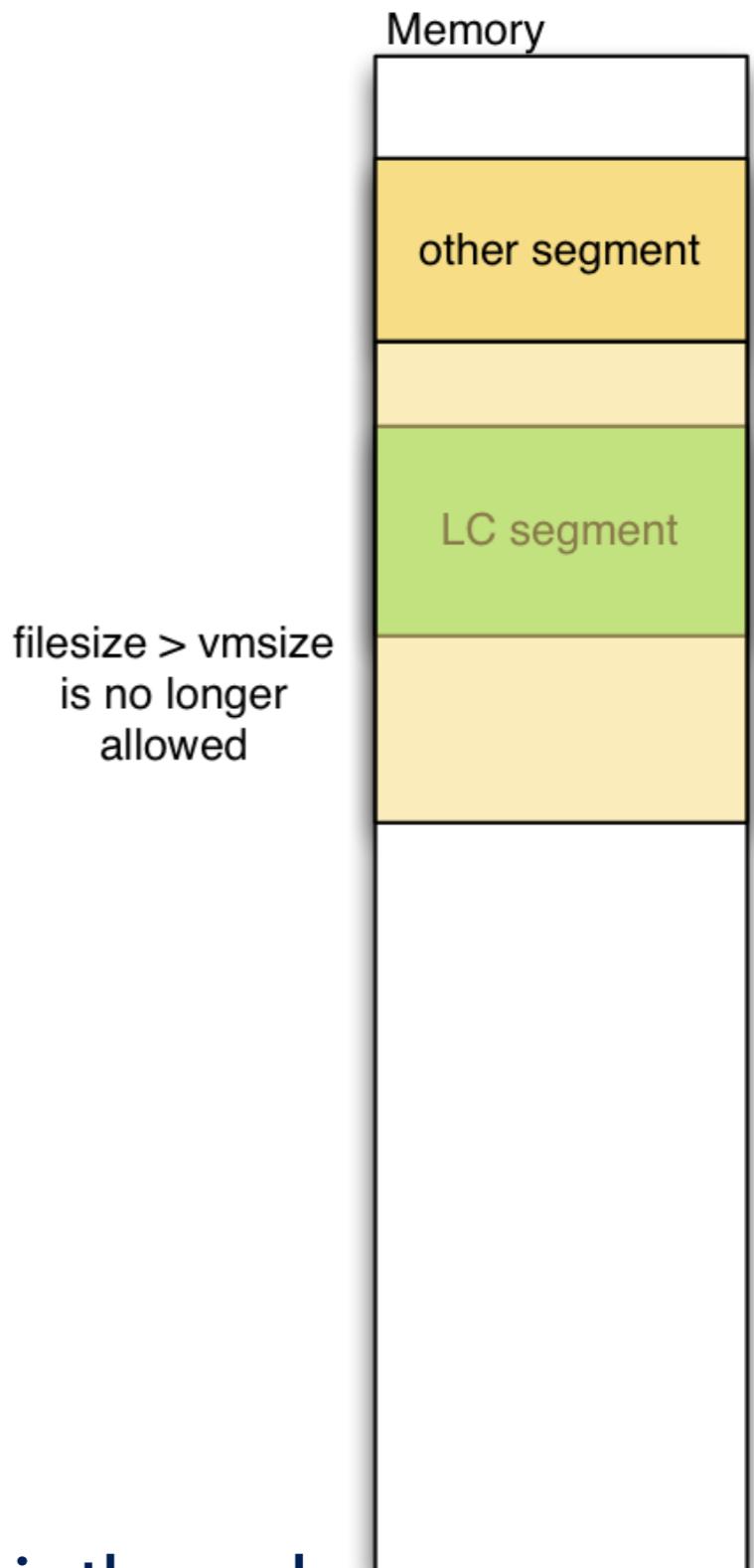
# Problem with Apple's third fix

- all these battles by Apple security with Chinese jailbreakers were for nothing
- the security checks by Apple all did detect overlaps by checking **vmaddr + vmsize**
- however **mapSegments** never used **vmsize**
- mapping into memory uses **filesize**
- **filesize** could be a lot larger than **vmsize**
- overlapping would never be detected



# Apple's fourth fix

- Apple now detects when **filesize** is bigger than **vmsize**
  - such segments are now forbidden
  - they now added a number of other checks that also killed another bypass by accident (will be disclosed in my blog)
  - it took Apple **only 2 years** to realise this ;-)
- **DISCLAIMER:** I am absolutely NOT claiming that this is the end



# **mach\_port\_kobject**



# mach\_port\_kobject

- debugging Mach API call
- returns **kobject** associated with a mach port
- **kobject** filled with different values depending on type of port
  - IOKit object ports: a kernel heap address containing a function table
  - Host ports: address of realhost variable in kernel
  - other ports: something else

```
*typep = (unsigned int) ip_kotype(port);
kaddr = (mach_vm_address_t)port->ip_kobject;
ip_unlock(port);

if (0 != kaddr && is_ipc_kobject(*typep))
    *addrp = kaddr;
else
    *addrp = 0;
```

- **SECURITY PROBLEM:** leaked addresses very useful for exploitation

# Fix 1

# mach\_port\_kobject in iOS 6.0

- Apple added kernel address obfuscation
- random 32 / 64 bit secret **vm\_kernel\_addrperm**
- added to all returned addresses
- should stop leakage of valid kernel pointers

```
*typep = (unsigned int) ip_kotype(port);
kaddr = (mach_vm_address_t)port->ip_kobject;
ip_unlock(port);

if (0 != kaddr && is_ipc_kobject(*typep))
    *addrp = VM_KERNEL_ADDRPERM(VM_KERNEL_UNSLIDE(kaddr));
else
    *addrp = 0;
```



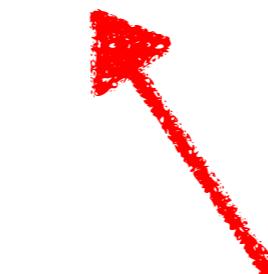
# Weakness of ADD Obfuscation

- all returned addresses are obfuscated with the same secret summand
  - **ADDR + SECRET = OBFUSCATED\_ADDR**
- a single pair of **ADDR** and its **OBFUSCATED\_ADDR** break the security
- if **SECRET** is known all obfuscated addresses can be easily decrypted

# Houston we have a problem ...

- Remember this?
  - **kobject** filled with different values depending on type of port
    - IOKit object ports: a kernel heap address containing a function table
    - **Host ports: address of realhost variable in kernel**
    - other ports: something else
- **SECURITY PROBLEM:**

we know realhost's address and can break obfuscation



we know  
plain address of  
this variable  
therefore we can break obfuscation

# Fix 2

# mach\_port\_kobject in iOS 8.0

- Apple now detects pointers inside kernel and does not obfuscate
- stops attack via e.g. host ports

```
*typep = (unsigned int) ip_kotype(port);
kaddr = (mach_vm_address_t)port->ip_kobject;
ip_unlock(port);

if (0 != kaddr && is_ipc_kobject(*typep))
    *addrp = VM_KERNEL_UNSLIDE_OR_PERM(kaddr);
else
    *addrp = 0;
```



# Houston we have a problem ...

- Remember this?
  - **kobject** filled with different values depending on type of port
    - IOKit object ports: a kernel heap address containing a function table
    - Host ports: address of realhost variable in kernel
    - **other ports: something else**
    - **master ports: the value of kobject is 1**
- **SECURITY PROBLEM:**  
Obfuscation secret is *mach\_port\_kobject(master)-1*

# Fun Fact

- master port attack was taught to **Pangu** in **SektionEins' iOS training**
- **Pangu** used it in their jailbreak in **June 2014**
- This means this attack was used in public **3 MONTHS** before iOS 8.0 release\*
- **TaiG** reused this trick in **November 2014** for their jailbreak

\* this means when fix 2 was released to public this was already known to be insufficient

# Fix 3

# mach\_port\_kobject in iOS 8.1.3

- Apple released **iOS 8.1.3** at end of January
- they fixed the problem **the Apple's way™**
- they just **removed the function altogether**
- removing the function **took them ONLY 7 MONTHS**
- This bug was assigned **CVE-2014-4496** and Apple **wrongly credits TaiG** for it

# kext\_request

# kext\_request()

- Mach API call
- allows applications to request information about kernel modules
- active operations are locked down (load, unload, start, stop, ...)
- passive operations partially working from even within the sandbox
- Apple **fixed it to unslide load addresses** to protect against **KASLR leaks**

# `kext_request()` - Get Loaded Kext Info

- of special interest is a sub request called
  - **Get Loaded Kext Info**
- returns a serialised dictionary with information about all loaded Kext
- information contained includes the mach-o headers
- Apple even modifies those headers to protect KASLR

# kext\_request() - Get Loaded Kext Info

```
mach_msg_type_number_t reqlen, resplen = 0, loglen = 0;
char *request, *response = NULL, *log = NULL;
kern_return_t kr;

request =
"<dict><key>Kext Request Predicate</key><string>Get Loaded Kext Info</string></dict>";

reqlen = strlen(request) + 1;

kext_request(mach_host_self(), 0, request, reqlen, &response,
&resplen, &log, &loglen, &kr);
```

# kext\_request() - iPhone 6plus / iOS 8.0.2

```
<dict ID="0"><key>_kernel__</key><dict ID="1"><key>OSBundleMachOHeaders</key><data ID="2">z/rt/gwAAAEEAAAAAAgAAAA  
8AAABACwAAAQAgAAAAAAZAAA0AEAAF9fVEVYVAAAAAIAACgP//wAgSAAAAAAAAAAAAAAIEgAAAAAAUAAAFA  
AAAAwAAAAAAABFx3R1eHQAAAAAAAX19URVhUAAAAAAwAAKA///rCJCAAAAAAAEAAADAAAAAAQAgAAAAAA  
AAAAAAAF9fY29uc3QAAAAAAABfx1RFWFQAAAAAAwFJCAoD//9oYAIAAAAAMAyQgAFAAAAAA  
AAAAAAAF9fY29uc3QAAAAAAAX19jc3RyaW5nAAAAAAAF9fVEVYVAAAAAAos0QCgP//2h  
+AwAAAAAKJNEAAAAAAIAAAAAAAZAAAyAIAAF9fREFUQAAAAAAQEGcGp//  
wBACwAAAAACBIAAAAAAAwAQAAAAAMAAAADAAAACAAAAABfx21vZF9pbml0X2Z1bmMAX19EQVRBAAAAAAABASAKA///  
CAIAAAAAAAIEgAAwAAAAAAACQAAAAAAAF9fbW9kX3R1cm1fZnVuYwBfx0RBVEEAAAAAAACEJIAoD///  
8AAgAAAAAAAgjSAADAAAAAAKAAAAAAAX19jb25zdAAAAAAAF9fREFUQAAAAAAQREgCgP//  
yCWAQAAAAAAECRIAQAAAAAAABfx2RhdGEAAAAAAAX19EQVRBAAAAAAASgKA///  
aNICAAAAAA4EkADgAAAAAAAF9fc2ZpX2NsYXNzX3J1ZwBfx0RBVEEAAAAAAaNJMAoD///  
8AAgAAAAAAAGiyTAADAAAAAAAX19zeXNjdGxfc2V0AAAAAF9fREFUQAAAAAABo1EwCgP//  
3AcAAAAAAALRMAAMAAAAAAABfx2JzcwAAAAAAAX19EQVRBAAAAAAATQKA///  
kF8GAAAAAAADAAAAAAQAAAAAAAF9fY29tbW9uAAAAAAABfx0RBVEEAAAAAAAGBTaoD///  
8YEQAAAAAAAMAAAAAAAGQAAACgCAABfx0tMRAAAAAAAIBTAoD///  
8AIAAAAAAADwTAAAAAAACAAAAAAwAAAAAAAX190Zxh0AAAAAAAF9fS0xEAAAAAAAgFMCgP//  
1ASAAAAAAAPBMAIAAAAAAAEAIAAAAAAAABfx2NzdHJpbmcAAAAAAAX19LTEQAAAAAAFCSUwKA///  
CaCAAAAAAABQAk0AAAAAAAGAAAAAAAF9fY29uc3QAAAAAAABfx0tMRAAAAAAAWJlTAoD///  
9oAAAAAAAFgJTQADAAAAAAAX19tb2RfaW5pdF9mdW5jAF9fS0xEAAAAAAADAmVMCgP//  
wgAAAAAAwA1NAAMAAAAAAkAAAAAAABfx21vZF90ZXJtX2Z1bmMAX19LTEQAAAAAAMiZUwKA///  
CAAAAAAAADICU0AAwAAAAACgAAAAAAAF9fYnNzAAAAAAABfx0tMRAAAAAAA0JlTAoD///  
8BAAAAAAABAAAAAAAGQAA0gAAABfx0xBU1QAAAAAAAKBTaoD///  
8AEAAAAAAQTQAAAAAAABAAAAAAwAAAAIAAAAAAAAX19tb2RfaW5pdF9mdW5jAF9fTEFTVAAAAAAoFMCgP//  
wgAAAAAAABBNAAMAAAAAAkAAAAAAABfx2xhc3QAAAAAAAX19MQVNUAAAAAAig8xKA///  
AAAAAAQAAAAAAQAAAAAAABkAACYAAAAX19QukVMSU5LX1RFWFQAAAwWQKA///  
AJDTAAAAAAoFIAAAAACQ0wAAAAAAwAAAABAAAAAAAF9fdGV4dAAAAAAABfx1BSRUXJTktfVEVYVAAAADBZAoD///  
8AkNMAAAAAACgUgAAAAAAAGQAAA0gAAABfx1BSRUXJTktfu1RBVEUAALDzEoD///  
8AAAAAAAGTQAAAAAAADAAAAwAAAAIAAAAAAAAX19rZXJuZWwAAAAAAAF9fUFJFTelOS19TVEFURQAsPMSgP//  
wAAAAAAACBNAAAAAAABfx2tleHRzAAAAAAAX19QukVMSU5LX1NUQRFAACw8xKA///  
AAAAAAIE0AAAAAAABkAACYAAAAX19QukVMSU5LX0l0Rk8AAADAzB0A///  
AFAIAAAAAAMCYBAAAAPhPCAAAAAAwAAAAMAAAABAAAAAAAF9faW5mbwAAAAAAABfx1BSRUXJTktfsU5GTwAAAMDME4D///  
4TwgAAAAAAwJgEAAAAAAAGQAAAEGAAABfx0xJTktfRELUAALDzEoD///  
94eQUAAAAAAgTQAAAAAeHkFAAAAABAAAAQAAAAAAAGAAABgAAADwnU8AnhAAANCnUACo8QEACwAAAFAAAAAAAACeEAAAn  
hAAAAAAACBNAPVHAAAbAAAAGAAAAPke8gf2KzpJlbujQoICCjclAAAAEAAAAAA  
AAAAAAWIANAoD//8AAAAAAACYAAAQAAAqF9PAEg+AAA=
```

```
</data><key>OSBundleCPUType</key><integer size="32" ID="3">0x10000c</integer><key>OSBundleCPUSubtype</key><integer size="32" ID="4">0x0</integer><key>CFBundleIdentifier</key><string ID="5">_kernel__</string><key>CFBundleVersion</key><string ID="6">14.0.0</string><key>OSBundleUUID</key><data ID="7">+R7yB/Yr0kmVu6NCggIKNw==</data><key>OSKernelResource</key><true/><key>OSBundleIsInterface</key><false/><key>OSBundlePrelinked</key><false/><key>OSBundleStarted</key><true/><key>OSBundleLoadTag</key><integer size="32" ID="8">0x0</integer><key>OSBundleLoadAddress</key><integer size="64" ID="9">0xfffffff8002002000</integer><key>OSBundleLoadSize</key>
```

# kext\_request() vs. Mark Dowd

I was informed by Mark Dowd  
that disclosure of bug  
was actually done by Tarjei Mandt.

So I am sorry for crediting it to the wrong person :)

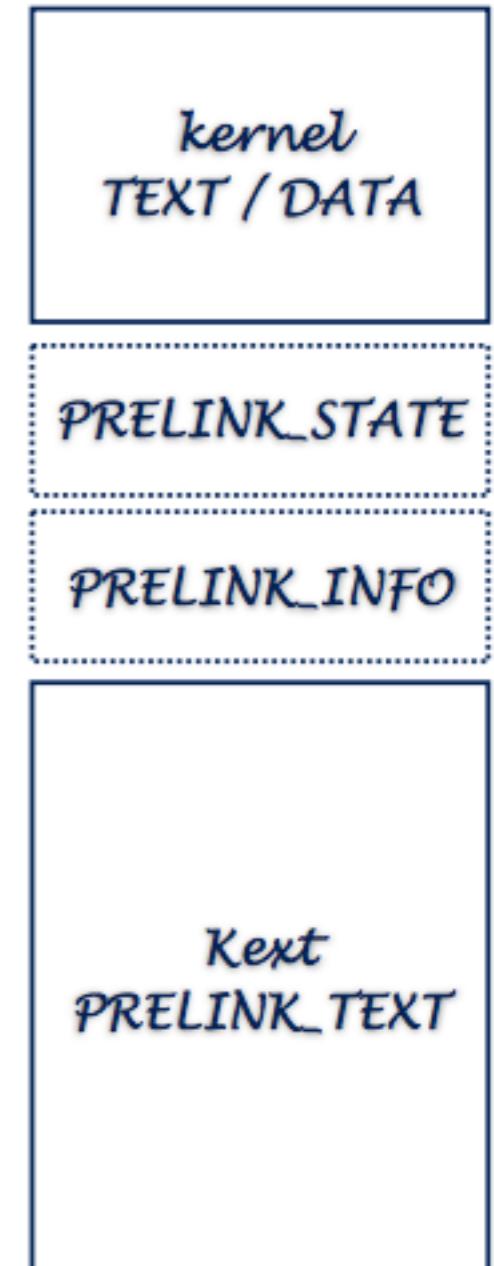


- so in October 2012 ~~Mark Dowd~~ disclosed that Apple forgot to unslide addresses in mach-o section definition
- Apple released a fix in iOS 6.0.1
- so KASLR was easy to defeat in iOS 6.0
- at this point everyone was auditing this function

# However ...

- however there were **ADDITIONAL** bugs
- macro Apple used did only unslide **main kernel** and **pre-loaded kernel extensions**
- but there are parts in the **kernelcache** that were in between and therefore not unslid
- leaks **KASLR** slide once again => **KASLR** broken

```
#define VM_KERNEL_IS_SLID(_o) \
    (((vm_offset_t)( _o ) >= vm_kernel_base) && \
     ((vm_offset_t)( _o ) < vm_kernel_top)) \
 \
#define VM_KERNEL_IS_KEXT(_o) \
    (((vm_offset_t)( _o ) >= vm_kext_base) && \
     ((vm_offset_t)( _o ) < vm_kext_top)) \
 \
#define VM_KERNEL_UNSLIDE(_v) \
    ((VM_KERNEL_IS_SLID(_v) || \
     VM_KERNEL_IS_KEXT(_v)) ? \
     (vm_offset_t)( _v ) - vm_kernel_slide : \
     (vm_offset_t)( _v ))
```



# Apple Releases a Fix

- end of January 2015 Apple releases iOS 8.1.3 to fix this
- **fixed it the Apple way: they remove the OSBundleMachOHeaders altogether**
- but they **lie about it in the release announcement** (claim it was fixed/not removed)
- took them only more than 2 years to realize this 2nd vulnerability Tarjei Mandt's
- and only 7 months from it being publicly used in a jailbreak
- people in training saw this within 10 min when checking out fix for Dowd's 2012 bug

- Kernel

Available for: iPhone 4s and later, iPod touch (5th generation) and later, iPad 2 and later

Impact: Maliciously crafted or compromised iOS applications may be able to determine addresses in the kernel

Description: An information disclosure issue existed in the handling of APIs related to kernel extensions. Responses containing an OSBundleMachOHeaders key may have included kernel addresses, which may aid in bypassing address space layout randomization protection. This issue was addressed by unsliding the addresses before returning them.

CVE-ID

CVE-2014-4491 : @PanguTeam, Stefan Esser

Tarjei Mandt's



# one more thing



# one more thing

- Of course all of these bugs also apply to OS X
- so no surprise that Apple fixed **kext\_request()** in OS X 10.10.2
- but for 10.10.2 they keep the **OSBundleMachOHeaders** and try to do it correctly

- Kernel

Available for: OS X Mountain Lion v10.8.5, OS X Mavericks v10.9.5, OS X Yosemite v10.10 and v10.10.1

Impact: Maliciously crafted or compromised applications may be able to determine addresses in the kernel

Description: An information disclosure issue existed in the handling of APIs related to kernel extensions. Responses containing an OSBundleMachOHeaders key may have included kernel addresses, which may aid in bypassing address space layout randomization protection. This issue was addressed by unsliding the addresses before returning them.

CVE-ID

CVE-2014-4491 : @PanguTeam, Stefan Esser

# one more thing

- at this point it might not surprise the audience anymore
- **kext\_request KASLR** info leak is still not fully fixed in Mac OS X 10.10.2
- the slide still leaks through the contained **OSBundleMachOHeaders**
- reason is that they use < instead of <= in the code
- therefore `__LAST::__last` (the last kernel symbol) is not unslid

```
#define VM_KERNEL_IS_SLID(_o)
    (((vm_offset_t)( _o ) >= vm_kernel_base) &&
     ((vm_offset_t)( _o ) <  vm_kernel_top)) \
#define VM_KERNEL_IS_KEXT(_o)
    (((vm_offset_t)( _o ) >= vm_kext_base) &&
     ((vm_offset_t)( _o ) <  vm_kext_top)) \
 \
#define VM_KERNEL_UNSLIDE(_v)
    ((VM_KERNEL_IS_SLID(_v) ||
      VM_KERNEL_IS_KEXT(_v)) ?
     (vm_offset_t)( _v ) - vm_kernel_slide :
     (vm_offset_t)( _v ))
```

# Jailbreak made in China



# Jailbreak made in China

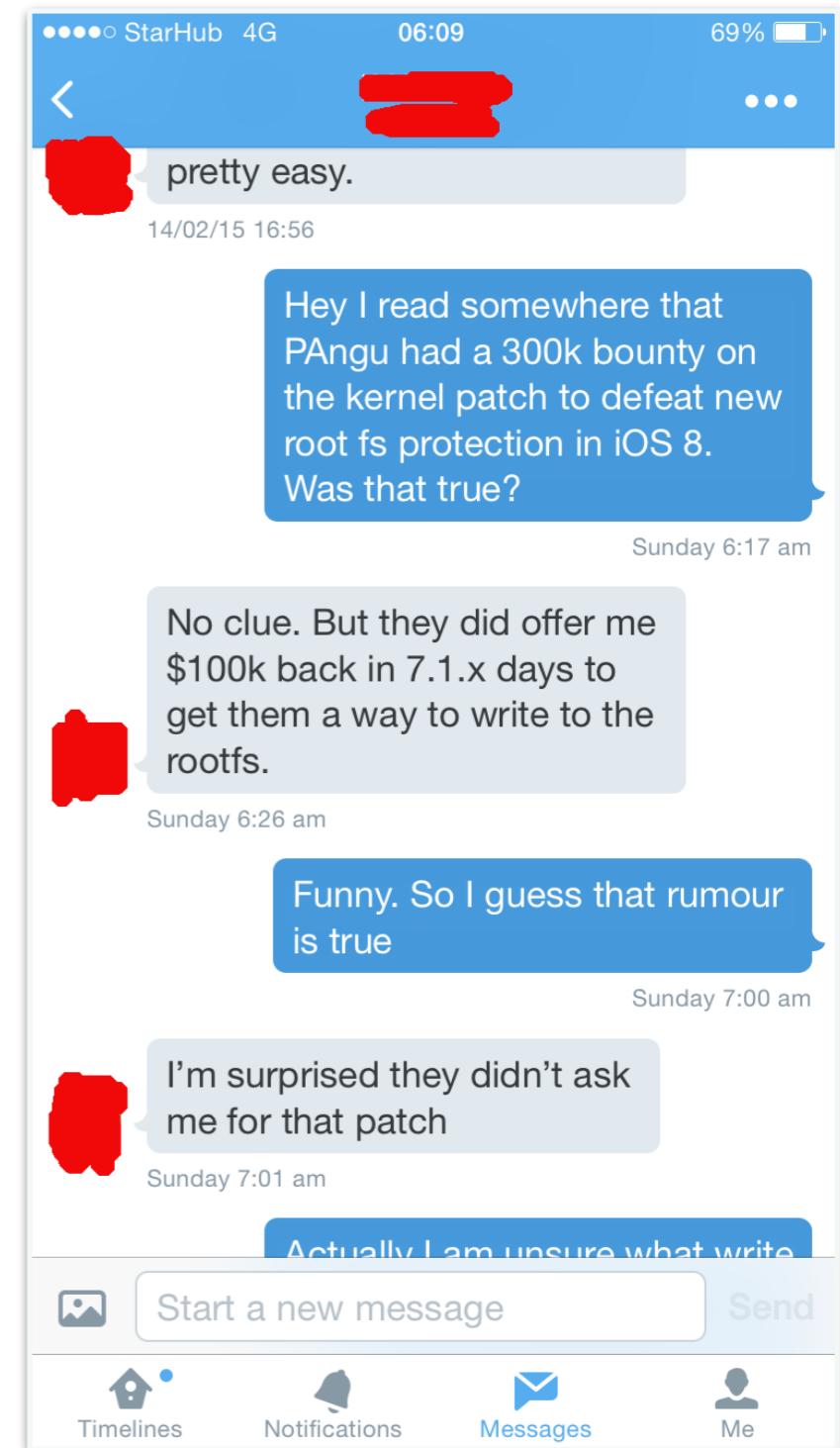
- there has been a change in guard
- recent Jailbreaks are no longer developed by "Westeners"
- now Chinese teams seem to rule the scene
- but why?

# Jailbreak made in China - Why?

- “Western” Jailbreak developers
  - shared a lot of tools and techniques
  - now have to deal with exploit export control regulations
  - worked hard for nearly nothing (few donations)
  - while information security companies made millions of their work
  - on top of that had to deal with greedy JB community

# Jailbreak made in China - Why?

- “Chinese” Jailbreak developers
  - are paid by Chinese companies with app stores ( e.g. 1 Mio USD )
  - use money to buy vulnerabilities / exploits ( from the west )
  - use questionable methods like stolen enterprise certificates
  - use public & non public code ignoring software licenses / copyright



at the moment I have no permission to disclose identity of jailbreaker who wrote this  
if you can offer 100.000 USD for buying a vulnerability / exploit this means you must get a lot for the end product

# Jailbreak made in China - and Apple

- so far “Chinese” Jailbreakers
  - had a lucky time because of Apple security’s total lack of QA
  - have mostly relied on techniques invented by westerners
  - could reuse the same vulnerabilities over and over

# Jailbreak made in China - Community

- so far “Chinese” Jailbreakers have
  - not released any code to advance the iOS research community
  - instead have heavily obfuscated their jailbreaks
  - intentionally removed patches to hinder work of other researchers
  - have destructive kernel patches that overwrite part of kernel binary
  - **they are in it for the money and don't want competition**

# Conclusion

# Conclusion

- looking at how Apple security fails over and over again shows they are not taking it seriously and have **no QA on their security fixes AT ALL.**
- Basically Apple Security's failed patches made the Chinese jb possible
- **IMHO** the fact that media is celebrating Jailbreaks instead of taking them as what they are: **exploits for vulnerabilities** is partly the reason why Apple does not take fixing them seriously
- media should stress the fact that every unfixed jailbreak makes it **CHEAP** for state sponsored attackers to spy on people
- actually it has been shown that several **iOS surveillance tools fully rely on release of public jailbreaks**

# Questions

