Modern Jailbreaking Techniques

A look behind the scenes on the post exploitation of the Dopamine jailbreak (iOS 15.0 - 15.4.1), including a comparison with previous jailbreaks

Lars Fröder (@opa334dev)

whoami

Lars Fröder / @opa334dev

- From <redacted>, Germany
- Started Programming in 2016
- Started Tweak Development in 2017
- Started Research into Jailbreaks in 2022 due to lack of iOS 15 Jailbreak
- Released TrollStore in 2022
- Released Dopamine Jailbreak (15.0 - 15.4.1) in 2023
- Currently employed @ Cellebrite and working on a Bachelor Thesis

Required Prerequisites

For modern Jailbreaks (arm64e, iOS 15.0+)

- Kernel Exploit
- Kernel PAC Bypass (Not a hard requirement for 15.2+)
- PPL Bypass
- Full control over Kernel Memory, excluding code pages
- With PAC bypass: Ability to call functions on kernel level
- All of these are provided by Fugu15 on 15.0 - 15.4.1 (thanks Linus)

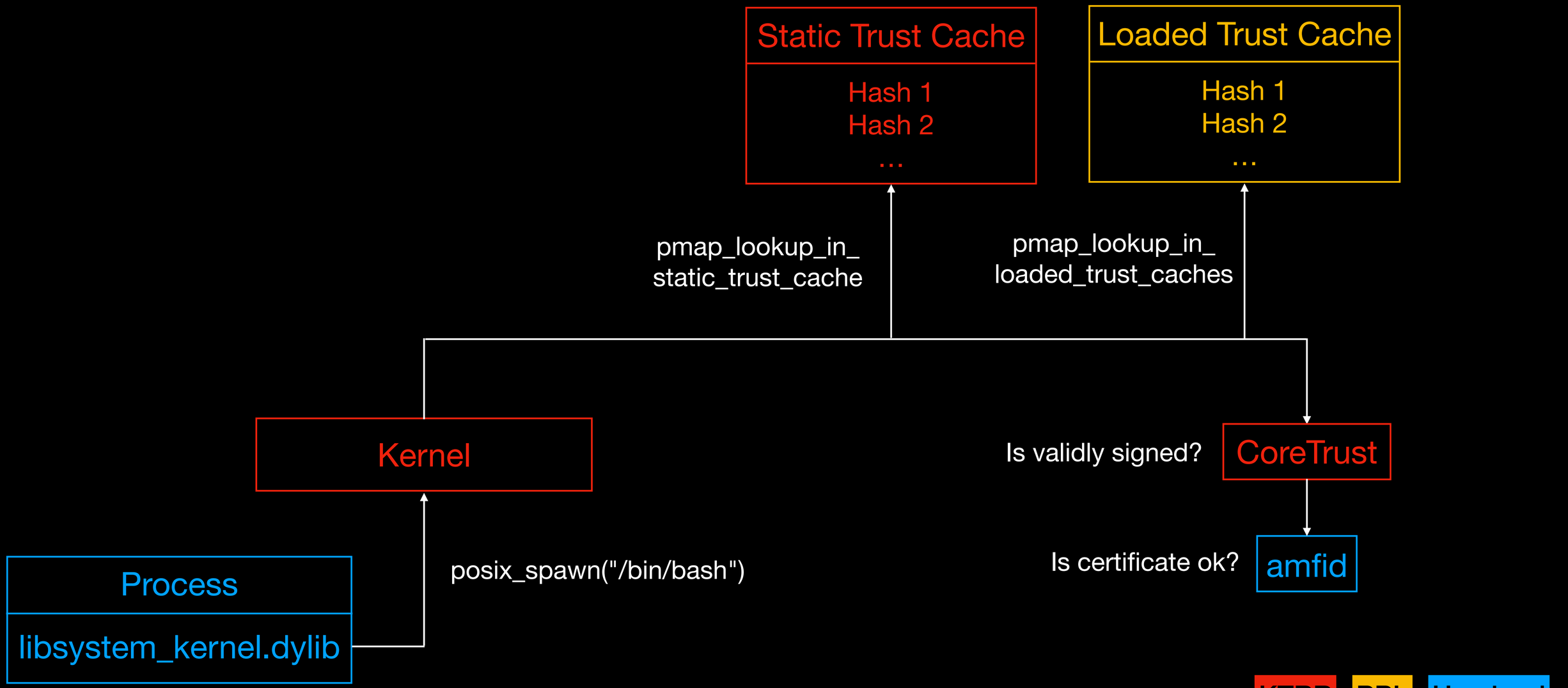
Goals of a Jailbreak

- Allow running unsigned binaries (Bypass Code Signing)
- Install bootstrap with dpkg environment and package manager app
- Allow system wide tweak injection
- Allow invalid pages so tweaks can actually hook stuff
- Do all of this while not impacting system stability 🥲

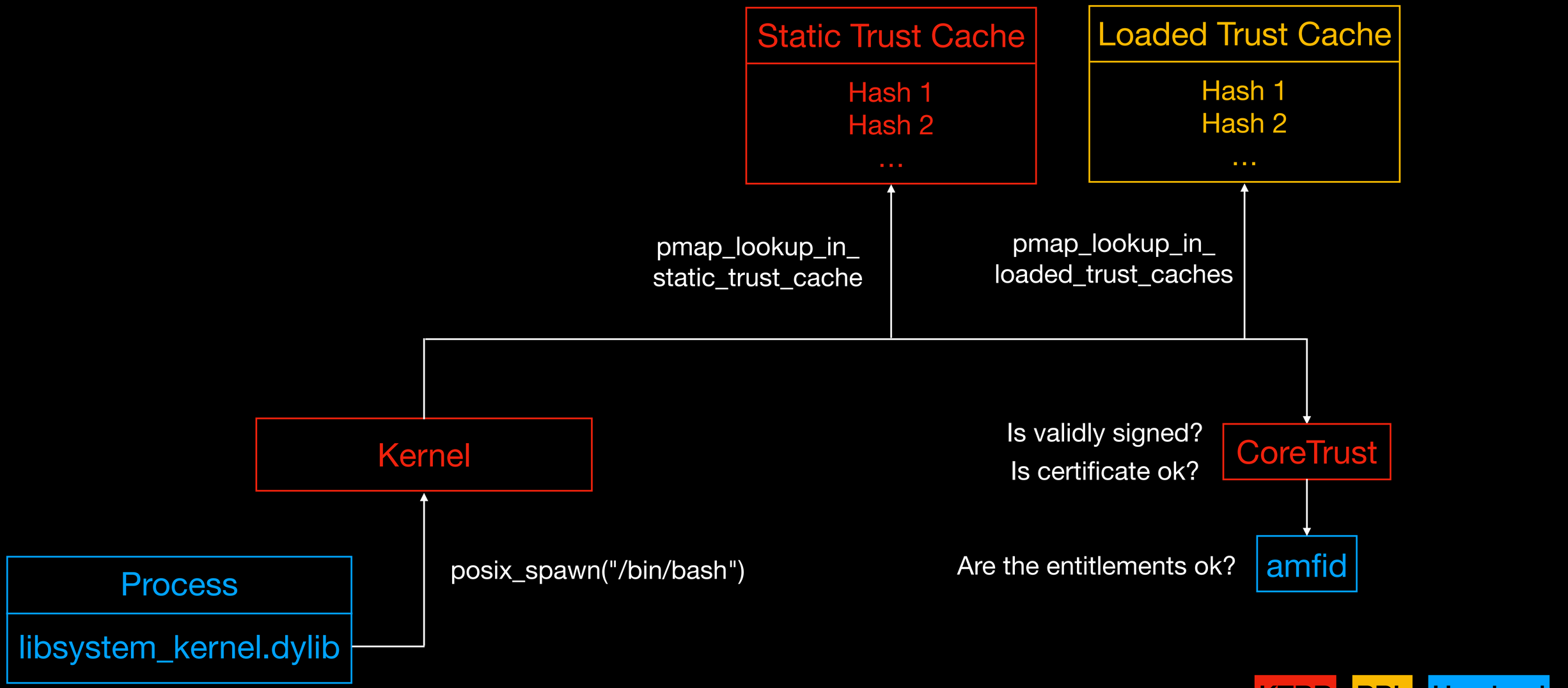
jailbreakd

- KTRRless idea: Move kernel hooks into userspace
- XPC service running in the background, started by the jailbreak
- Alternative to Kernel Text Patches
- Works together with system wide hook to emulate all „classic“ jailbreak functionality
- Handles all operations that require the use of exploit primitives

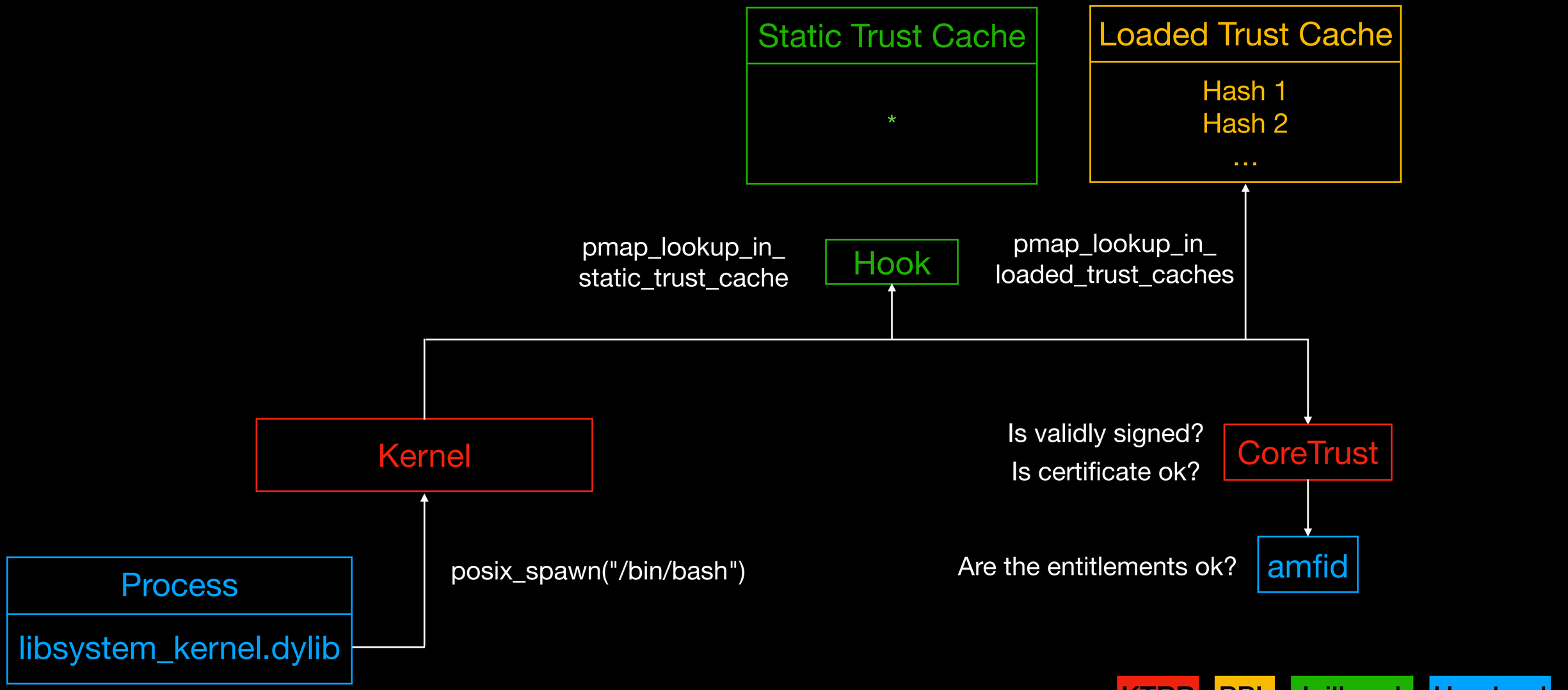
Code Signing (iOS <=14)



Code Signing (iOS >=15)

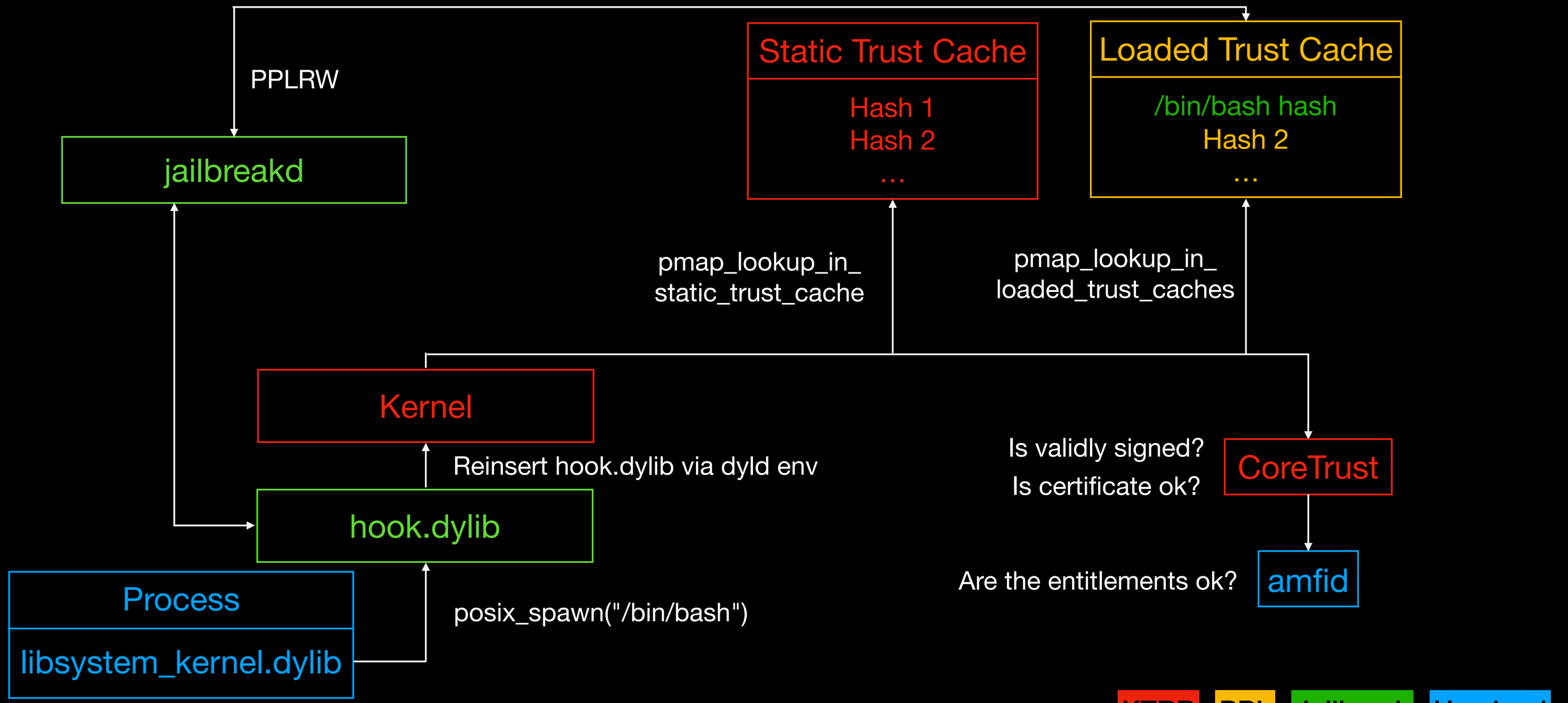


Code Signing Bypass (With KTRR Bypass)

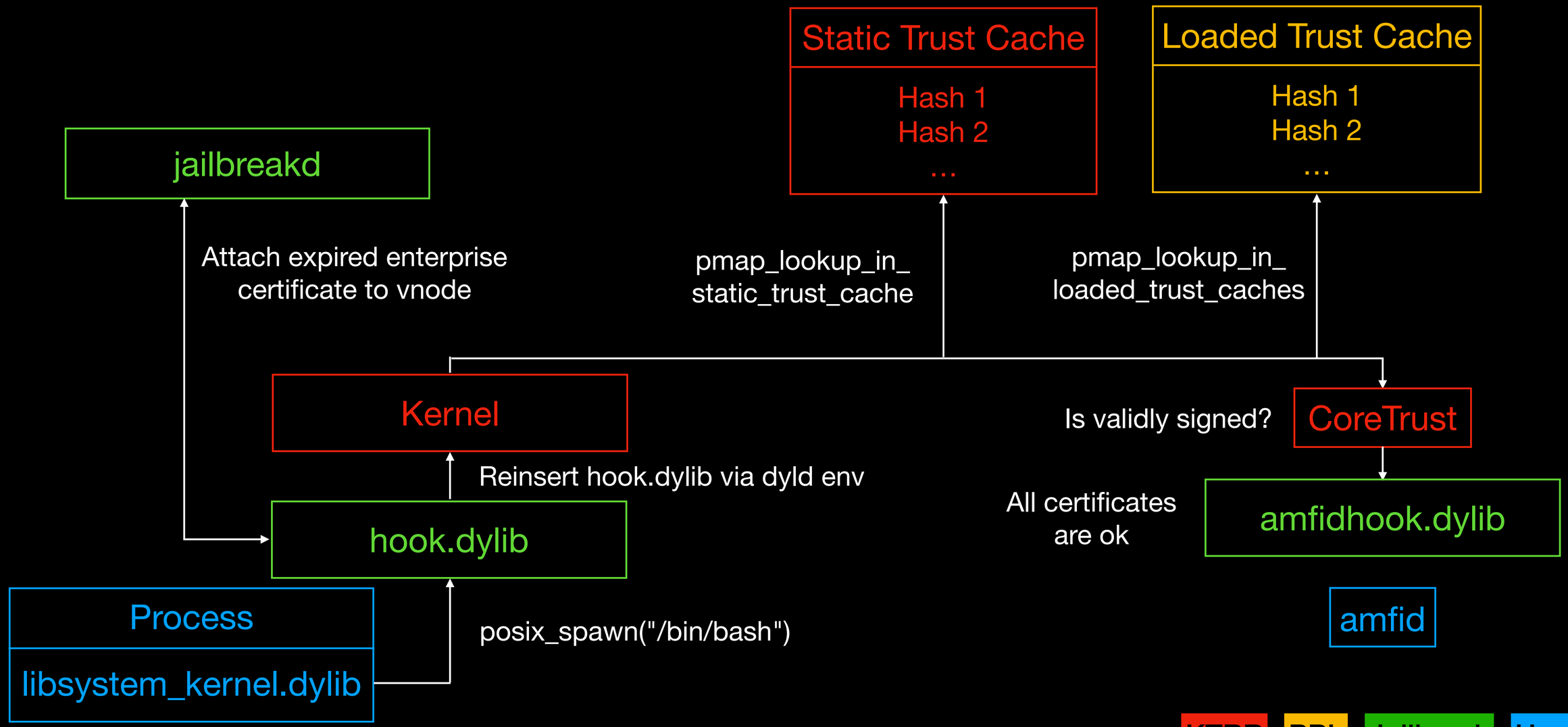


KTRR PPL Jailbreak Userland

Code Signing Bypass (With PPL Bypass)



Code Signing Bypass (Only KRW, iOS <=14)



Library Insertion Check

DYLD_INSERT_LIBRARIES="/usr/lib/systemhook.dylib"

dyld

```
    const uint64_t amfiFlags = getAMFI(process, syscall);
    this->allowAtPaths              = (amfiFlags & AMFI_DYLD_OUTPUT_ALLOW_AT_PATH);
    this->allowEnvVarsPrint         = (amfiFlags & AMFI_DYLD_OUTPUT_ALLOW_PRINT_VARS);
    this->allowEnvVarsPath          = (amfiFlags & AMFI_DYLD_OUTPUT_ALLOW_PATH_VARS);
    this->allowEnvVarsSharedCache   = (amfiFlags & AMFI_DYLD_OUTPUT_ALLOW_CUSTOM_SHARED_CACHE);
    this->allowClassicFallbackPaths = (amfiFlags & AMFI_DYLD_OUTPUT_ALLOW_FALLBACK_PATHS);
    this->allowInsertFailures       = (amfiFlags & AMFI_DYLD_OUTPUT_ALLOW_FAILED_LIBRARY_INSERTION);
    this->allowInterposing          = (amfiFlags & AMFI_DYLD_OUTPUT_ALLOW_LIBRARY_INTERPOSING);
    this->allowEmbeddedVars         = (amfiFlags & AMFI_DYLD_OUTPUT_ALLOW_EMBEDDED_VARS);
#if TARGET_OS_SIMULATOR
    this->allowInsertFailures       = true; // FIXME: amfi is returning the wrong value for simulators <rdar://74025454>
#endif
```

Kernel

```
__int64 __fastcall _check_dyld_policy_internal(struct proc *a1, __int64 *a2)
{
  __int64 v4; // x21
  __int64 result; // x0

  if ( proc_issetugid(a1) )
  {
    v4 = 0x49LL;
  }
  else if ( (proc_has_get_task_allow(a1) & 1) != 0 )
  {
    v4 = 0x5FLL;
  }
  else if ( (unsigned int)proc_has_entitlement(a1, "com.apple.developer.swift-playgrounds-app.development-build") )
  {
    v4 = 0x5BLL;
  }
  else
  {
    v4 = 0x49LL;
  }
  result = cs_require_lv(a1);
  *a2 = v4 | (32LL * ((_DWORD)result != 0));
  return result;
}
```

Library Insertion
Possible Solutions

- Satisfy Kernel Check
  - Inject get-task-allow entitlement
  - Patch proc structure to have CS_GET_TASK_ALLOW csflag
- Patch dyld
  - Make Kernel find different dyld file
  - Patch getAMFI to always return the desired flags

Bind Mount

- Mount directory (or file) on top of another directory
- E.g. mount /var/jb/basebin/.fakelib on top of /usr/lib
- When accessing /usr/lib, the original contents are invisible and the contents of /var/jb/basebin/.fakelib are shown
- Path to mount on top of restricted by sandbox
- iOS only supports Read-Only Bind Mounts (Hard Limitation)

Bind Mount Freely on iOS

- Get kernel credentials
- Overwrite credentials of current process with kernel credentials
- Perform bind mount regularly via „mount“ syscall
- Restore original credentials
- Profit! (Still read only)
- Can be used to mount directory containing patched „dyld“ on top of „/usr/lib“

Dyld Patch

```
; __int64 __fastcall dyld4::ProcessConfig::Security::getAMFI
__ZN5dyld413ProcessConfig8Security7getAMFIERKNS0_7ProcessERN
                                        ; CODE XREF: dyld4::

var_28          = -0x28
var_24          = -0x24
var_20          = -0x20
var_10          = -0x10
var_s0          =  0

                PACIBSP
                SUB             SP, SP, #0x40
                STP             X22, X21, [SP,#0x30+var_20]
                STP             X20, X19, [SP,#0x30+var_10]
                STP             X29, X30, [SP,#0x30+var_s0]
                ADD             X29, SP, #0x30
                MOV             X20, X2
                MOV             X21, X1
                MOV             X19, X0
                LDR             X0, [X1] ; this
```

```
; dyld4::ProcessConfig::Security::getAMFI
__ZN5dyld413ProcessConfig8Security7getAMF
                                        ;
                MOV             X0, #0xDF
                RET
```

- Resign patched file using Security.framework
- Calculate new cdhash and upload to Loaded Trust Cache via jailbreakd

Library Insertion Solution

- Static dyld patch
- Replace original dyld file via bind mount
- System library insertion forever*
- *Downside: truly forever, bind mount can only be reverted by reboot
- Very simple and deterministic, 100% reliable, no room for failure

System Wide Code Injection

Using systemhook.dylib

- Defeats code signing
- Loosens sandbox via sandbox extensions
- Handles setuid bits (Normally ignored, needs fixup by jailbreakd)
- Makes Jetsam less strict
- If necessary, enables invalid code pages
- Loads Tweak Loader library if it exists

Allowing Invalid Code Pages
For Function Hooks

- Normally done by ptrace syscall, which invokes cs_allow_invalid in kernel
- Problem: cs_allow_invalid has PPL level check whether the target process has the get-task-allow entitlement
- cs_allow_invalid, amongst other things, sets the wx_allowed bit on the pmap structure of the process to true
- Solution: Find process structure in kernel memory, find associated pmap, set wx_allowed to true using PPLRW
- (Additionally: Hook ptrace to always work by delegating it to jailbreakd)

opainject

For kickstarting system wide injection

- Gets task port of a process
- Creates new thread in task
- Makes arbitrary thread call dlopen with a specified path
- Waits for return value
- Profit???

Launchd Hook

- Reuses posix_spawn Hook from systemhook.dylib to insert it into everything spawned by it
- sandbox_check hook to allow certain types of IPC unconditionally (cy: or lh: prefix)
- Load third party launch daemons
- Hook XPC handler with functionality for jailbreakd to get back primitives after it restarts
- Mother of all primitives, as the system lives and dies with it

Boomerang

Preserving Jailbreak through Userspace Reboots

- Userspace reboots are either triggered by the user or randomly by the system
- We have no other option than to support them
- Userspace reboot: kill(-1, 9), exec("/sbin/launchd")
- Hook exec, spawn boomerang process, make it detach from launchd
- launchd sends PPLRW and kcall primitives to boomerang
- Call original exec implementation
- In dylib constructor of launchdhook.dylib in new launchd, contact boomerang to retrieve back primitives, then reapply all hooks

Apple vs Fork

- In iOS 15, Apple, probably on accident, broke fork for any binary not in TrustCache
- Child process cannot map any binary not in TrustCache, does not affect libraries (or us, because we use TrustCache injection)
- Child process does not inherit wx_allowed
- All pages that have hooks applied loose their executable permission
- Problem: fork is used extensively in bootstrap (e.g. in bash, apt, dpkg, …)

forkfix

- It’s own library, to make sure none of the pages in it become invalid
- Hook __fork, call our own reimplementation of it instead
  - Pause child process
  - Parent calls out to jailbreakd
    - Set wx_allowed
    - Copy protection flags of all vm_entries from parent process to child process
  - Resume child process

To Summarize

- Spawn jailbreakd, handoff primitives
- Inject library into launchd, that gets primitives from jailbreakd
- Trigger userspace reboot (Boomerang preserves primitives)
- Insert systemhook.dylib into all spawned processes
- systemhook.dylib reinserts itself into all binaries spawned from processes that it’s already injected in
- Install Bootstrap, Package Manager, Profit!

The End

Questions?