AMFI Launch Constraints - First Quick Look

June 14, 2022

4 minutes read

Dropping some initial quick notes for a new security feature I ran into on macOS Ventura. It's called "Launch Constraints" and lives inside AMFI.

Do the following experiment: Copy `Terminal.app` to your HOME folder and try to run it on Monterey and Ventura. On the former it will work without any issues, on the other it will fail, and we will get the following error:

```
2022-06-14 05:59:55.254678+0200 0x5481     Default     0x0
kernel: (AppleMobileFileIntegrity) AMFI: Launch Constraint Violation (enfor
info: c[1]p[1]m[1]e[2], (Constraint not matched) launching proc[vc: 1 pid:
/Users/ace/Terminal.app/Contents/MacOS/Terminal, launch type 0, failure pro
1112]: /Users/ace/Terminal.app/Contents/MacOS/Terminal
```

This is interesting. If we copy it to other locations, it will fail as well. The reason looks to be something called "Launch Constraint". So it seems that it will disallow the execution of system apps outside their location. This is pretty nice, as this has been abused plenty of times in the past, as moving them out of their place usually allowed someone inject code into them. Remember these TCC bypasses?

Change home directory and bypass TCC aka CVE-2020-27937 Bypassing MacOS Privacy Controls

But there is more!

If we try to execute any of the binaries from Terminal, which has an

associated LaunchDaemon / Agent, it will fail.

```
ace@ventura ~ % /usr/libexec/periodic-wrapper
zsh: killed     /usr/libexec/periodic-wrapper
```

and:

```
2022-06-14 06:09:24.419580+0200 0x65f4     Default     0x0
kernel: (AppleMobileFileIntegrity) AMFI: Launch Constraint Violation (enfor
info: c[1]p[1]m[1]e[1], (Constraint not matched) launching proc[vc: 1 pid:
/usr/libexec/periodic-wrapper, launch type 0, failure proc [vc: 1 pid: 1195
/usr/libexec/periodic-wrapper
```

Also nice, this has been also abused in the past. Remember powerdir?

[New macOS vulnerability, “powerdir,” could lead to unauthorized user data access - Microsoft Security Blog]

Even if you copy these binaries out of their location, they will fail.

Let’s take a quick look into AMFI. (Since the KEXT binary is no longer present, you need either the KDK or load it from the kernel cache). It has some interesting new functions:

```
ConfigurationSettings::enforceLaunchConstraints(void)
ConfigurationSettings::allow3rdPartyLaunchConstraints(void)
_proc_check_launch_constraints(proc *,int,int,void *,ulong,launch_constrain
```

The settings are initialized in:

```
char configurationSettingsInit(void)
```

```
{
  IORegistryEntry *v0; // r12
  const OSMetaClassBase *v1; // rax
  OSMetaClassBase *v2; // rax
  boolean_t v3; // ebx
  int v4; // r13d
  char *v5; // rax
  int v6; // r15d
  int v7; // r14d
  bool v8; // zf
  bool v9; // bl
  bool v10; // r15
  boolean_t v12; // [rsp+Ch] [rbp-34h]
  int v13; // [rsp+10h] [rbp-30h]
  int v14[11]; // [rsp+14h] [rbp-2Ch] BYREF

  sysctl_register_oid(&sysctl__security_mac_amfi);
  sysctl_register_oid(&sysctl__security_mac_amfi_developer_mode_status);
  sysctl_register_oid(&sysctl__security_mac_amfi_developer_mode_resolved);
  sysctl_register_oid(&sysctl__security_mac_amfi_launch_constraints_enforce
  sysctl_register_oid(&sysctl__security_mac_amfi_launch_constraints_3rd_par
  sysctl_register_oid(&sysctl__security_mac_amfi_launch_env_logging);
  v0 = IORegistryEntry::fromPath("/chosen", gIODTPlane, 0LL, 0LL, 0LL);
  if ( v0 )
  {
    v1 = (const OSMetaClassBase *)((__int64 (__fastcall *)(IORegistryEntry
                                    v0,
                                    "security-mode-change-enable");
    v2 = OSMetaClassBase::safeMetaCast(v1, OSData::metaClass);
    if ( v2 )
    {
      if ( *(_DWORD *)((__int64 (__fastcall *)(OSMetaClassBase *))v2->__vft
        bootedWithModeChangeEnabled = 1;
    }
  }
  macOSPolicyConfigurationInit();
  v14[0] = 0;
  v3 = PE_parse_boot_argn("amfi_enforce_launch_constraints", v14, 4);
```

```
v4 = v3;
if ( v3 )
  v4 = v14[0];
v12 = PE_parse_boot_argn("amfi_allow_3p_launch_constraints", v14, 4);
v13 = v14[0];
LODWORD(v5) = PE_parse_boot_argn("BATS_TESTPLAN_ID", v14, 4);
v6 = (int)v5;
v7 = v14[0];
v8 = v3 == 0;
v9 = v3 != 0;
if ( v8 || !v4 )
{
  LODWORD(v5) = csr_check(2LL);
  if ( (_DWORD)v5 )
  {
    LODWORD(v5) = csr_check(&dword_10);
    if ( !(_DWORD)v5 )
    {
      v5 = &BootedDevice;
      if ( BootedDevice
        || (v10 = v6 != 0,
            LOBYTE(v5) = macOSPolicyConfig::bniAllowAsPlatform((macOSPoli
            (_BYTE)v5)
        || v7 != 0 && v10
        || (LOBYTE(v5) = v4 == 0, v4 == 0 && v9) )
      {
        launchConstraintsEnforced = 0;
      }
      if ( v12 && v13 )
        launchConstraint3rdPartyAllowed = 1;
    }
  }
  else
  {
    launchConstraintsEnforced = 0;
  }
}
else
```

```
  {
    launchConstraintsEnforced = 1;
  }
  if ( v0 )
    LOBYTE(v5) = ((__int64 (__fastcall *)(IORegistryEntry *))v0->release_0)
  return (char)v5;
}
```

We find two things. One is that the settings are taken from the PE boot_args and that there is an associated sysctl entry. We can find that:

```
ace@ventura ~ % sysctl security | grep launch_constraints
security.mac.amfi.launch_constraints_enforced: 1
security.mac.amfi.launch_constraints_3rd_party_allowed: 0
```

`_proc_check_launch_constraints` does the heavy lifting, and it will need further reversing, as it's a long function. It will be interesting to see how it works in the deep.

If SIP is disabled this feature won't be enforced:

```
if ( ConfigurationSettings::launchEnvDebugLoggingEnabled((ConfigurationSett
{
  v17 = 0;
  IOLog("SIP is disabled, not checking launch constraints\n");
}
```

I think this feature mitigates a whole exploit class and I think this was long overdue by Apple. So good job to everyone who was involved! Regardless it's nice to see this in work, even if my job is harder now :D