Customizing ART virtual machine of AOSP

After following the steps below to modify the AOSP source code, the compiled Android system will record Java function calls and JNI calls during the app's runtime.

These codes were not created by us. We obtained them indirectly from a blog on the internet. We would like to express our gratitude to the author of the code (although we are unable to determine who the original author is, but if you happen to come across this, we sincerely thank you).

1. Tracing Java method calling

Add the following code at the beginning of the `PerformCall` function in the file `art/runtime/common_dex_operations.h`.

```
// add
ArtMethod* callee = callee_frame->GetMethod();
std::ostringstream oss;
oss << "[PerformCall] " << caller_method->PrettyMethod() << " --> " << callee->PrettyMethod();
if(strstr(oss.str().c_str(),"PerformCallBefore")){
 LOG(ERROR) << oss.str();
}
// add
```

2. Tracing JNI function calling

Add the following code in the `InvokeWithArgArray` function in the file `art/runtime/reflection.cc`.

```
// add
ArtMethod* artMethod = nullptr;
Thread* self = Thread::Current();
const ManagedStack* managedStack = self->GetManagedStack();
if(managedStack != nullptr) {
    ArtMethod** tmpArtMethod = managedStack->GetTopQuickFrame();
    if(tmpArtMethod != nullptr) {
        artMethod = *tmpArtMethod;
    }
}
if(artMethod != nullptr) {
    std::ostringstream oss;
    oss << "[InvokeWithArgArray before] " << artMethod->PrettyMethod() << " --> "<< method->PrettyMethod();
    if(strstr(oss.str().c_str(),"InvokeWithArgArrayBefore")){
        LOG(ERROR) << oss.str();
    }
}
// add

```

3. Enable force interpret mode

Add the following code in the end of the file `art/runtime/interpreter/interpreter.cc`.

```
// add
extern "C" void forceInterpret(){
    Runtime* runtime = Runtime::Current();
    runtime->GetInstrumentation()->ForceInterpretOnly();
    LOG(WARNING) << "forceInterpret is called";
}
// add

```

4. Specify interpreter

Modify the value of the variable `kInterpreterImplKind` in the file `art/runtime/interpreter/interpreter.cc`.

```
kInterpreterImplKind = kSwitchImplKind
```

5. Tracing Smali instructions

Add the following code in the file `art/runtime/interpreter/interpreter_switch_impl-inl.h`:

Before `while true`:

```
// add
bool shouldTrace = false;
if(strstr(shadow_frame.GetMethod()->PrettyMethod().c_str(), "ExecuteSwitchImplCppBefo
re")) {
    shouldTrace = true;
}
// add

```

In `while(true)` loop:

```
// add
// TraceExecution(shadow_frame, inst, dex_pc);
if (shouldTrace) {
    myTraceExecution(shadow_frame, inst, dex_pc);
}
// add

```

Add the following code in the end of the file `art/runtime/interpreter/interpreter_common.h`:

```
// add
static inline void myTraceExecution(const ShadowFrame& shadow_frame, const Instructio
n* inst,
                                    const uint32_t dex_pc) REQUIRES_SHARED(Locks::mut
ator_lock_) {
```

```
    std::ostringstream oss;
    oss << "[FuncName] " << shadow_frame.GetMethod()->PrettyMethod() << "\t"
        << android::base::StringPrintf("[Address] 0x%x: ", dex_pc)
        << inst->DumpString(shadow_frame.GetMethod()->GetDexFile()) << "\t[Regs]";
    for (uint32_t i = 0; i < shadow_frame.NumberOfVRegs(); ++i) {
        uint32_t raw_value = shadow_frame.GetVReg(i);
        ObjPtr<mirror::Object> ref_value = shadow_frame.GetVRegReference(i);
        oss << android::base::StringPrintf(" vreg%u=0x%08X", i, raw_value);
        if (ref_value != nullptr) {
            if (ref_value->GetClass()->IsStringClass() &&
                !ref_value->AsString()->IsValueNull()) {
                oss << "/java.lang.String \"" << ref_value->AsString()->ToModifiedUtf
8() << "\"";
            } else {
                oss << "/" << ref_value->PrettyTypeOf();
            }
        }
    }
    if(strstr(oss.str().c_str(), "myTraceExecutionBefore")) {
        LOG(ERROR) << oss.str().c_str();
    }
}
// add

```

6. How to use

Based on the Frida environment, running the following hook script on a computer can print out the tracking record.

```

function trace() {
    var libcModule = Process.getModuleByName("libc.so");
    var strstrAddr = libcModule.getExportByName("strstr");
    Interceptor.attach(strstrAddr, {
        onEnter: function (args) {
            this.arg0 = ptr(args[0]).readUtf8String();
            this.arg1 = ptr(args[1]).readUtf8String();

            if (this.arg1.indexOf("InvokeWithArgArrayBefore") != -1) { //print JNI ca
lling
                console.log("JNI_CALL"+ "---" + this.arg0);
            }

            if (this.arg1.indexOf("PerformCallBefore") != -1) { //print JAVA method c
alling
                console.log("JAVA_CALL" + "---" + this.arg0);
            }

            if (this.arg1.indexOf("myTraceExecutionBefore") != -1) { // print smali i
nstructions
                console.log("TRACE_SMALI" + "---" + this.arg0);
```

```
            }

        }, onLeave: function (retval) {
            if (this.arg1.indexOf("ExecuteSwitchImplCppBefore") != -1) {
                retval.replace(1);
            }
        }
    });
}
```