Reversing Android RASP

Analyzing RASP checks with radare2 & frida

> R2CON_2025


$ whoami
Name: Govind Sharma (@apkunpacker)
Role: Mobile Security Researcher
Socials: Twitter | GitHub | LinkedIn

THE CHALLENGE

Modern RASP Solutions Are Getting Smarter

Native libraries employ advanced obfuscation, runtime decryption, and memory protection schemes that make traditional static analysis ineffective.

🔐 Encrypted Code

ELF sections encrypted until runtime execution

🛡️ Memory Protection

Dynamic mmap/mprotect permission changes

⚡ Anti-Debug

Advanced detection and prevention mechanisms

TECHNIQUE #1

.init_array Hook

```
$ radare2 -A librasp.so
[0x00000000]> iS~.init_array
8 0x00002b00 0x8 0x00006b00 0x8 -rw- 0x3 INIT_ARRAY
.init_array
[0x00000000]>
```

 # Constructor Magic

Functions execute before JNI_OnLoad automatically

 # Auto-Decrypt

Decryption routines trigger on dlopen() call

 # Memory Dump

Capture decrypted code from runtime memory

ATTACK FLOW

Load Library

↓

Init Array Fires

↓

Code Decrypts

↓

Dump Memory

Usages

Start

↓

Compute size of target sections .data

↓

Load library via `dlopen()`

↓

Automatic execution of .init_array

↓

Decryption occurs in memory (no full app launch)

↓

Extract decrypted strings from process memory

↓

Finish

Limitations

> Require proper environment to load libraries (DT_NEEDED)

> Not all RASP decrypt everything with init_array

▶ LIVE DEMO

`init_array Exploitation`

TECHNIQUE #2

Memory Surveillance

- **mmap Watch**
  Track anonymous memory allocations
- **mprotect Hook**
  Capture permission elevation events
- **Auto Dump**
  Extract executable memory regions

? WHY THIS WORKS

RASP must make memory executable before running decrypted code.

> *By hooking mprotect, we catch the exact moment code becomes live.*

```
var file_path = "/data/data/com.r2con.demo/random.so";
Interceptor.attach(
  Module.findExportByName("libc.so", "mprotect"), {
  onEnter: function(args) {
    this.address = args[0];
    this.len = args[1];
    this.protection = args[2];
    if (this.protection == 0x5) {
      this.hook = true;
      console.warn("mprotect : ",
        args[0], args[1], args[2]);
    }
  },
  onLeave: function(retval) {
    if (this.hook) {
      var file_handle = new File(file_path, "wb");
      var len = this.len.toInt32()
      var buffer = this.address.readByteArray(len);
      file_handle.write(buffer);
      file_handle.flush();
      file_handle.close();
      console.warn("Dump : ", file_path);
    }
  },
});
```

Memory Protection & RASP

> **mmap():** Allocates anonymous memory regions

> **mprotect():** Changes page permissions → "rx"

> **mprotect():** Maps encrypted code, executes in protected pages

> **Nasty checks:** All RASP logic in newly mapped regions

> Standard tools miss dynamically created code

Usages

Start

↓

Locate `mprotect()`

↓

Track usage

(Analyze cross-refs & memory protections)

↓

Track usage

(Imports/offsets & memory protections)

↓

Hook & Monitor

(Monitor new memory regions)

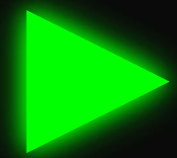
↓

Dump

(New memory regions to uncover hidden code)

↓

Finish



LIVE DEMO

Memory Surveillance in Action

THANK YOU

Questions?

> Keep reversing, keep learning

> R2CON_2025