嵌入式系统联谊会
www.esbf.org

梆梆安全
BANGCLE

基于IR码的IoT源码漏洞分析

梆梆安全

陈忠

2019年10月

稳 如 泰 山 · 值 得 托 付

2019年中国上海嵌入式系统安全论坛

梆梆安全
BANGCLE

议题

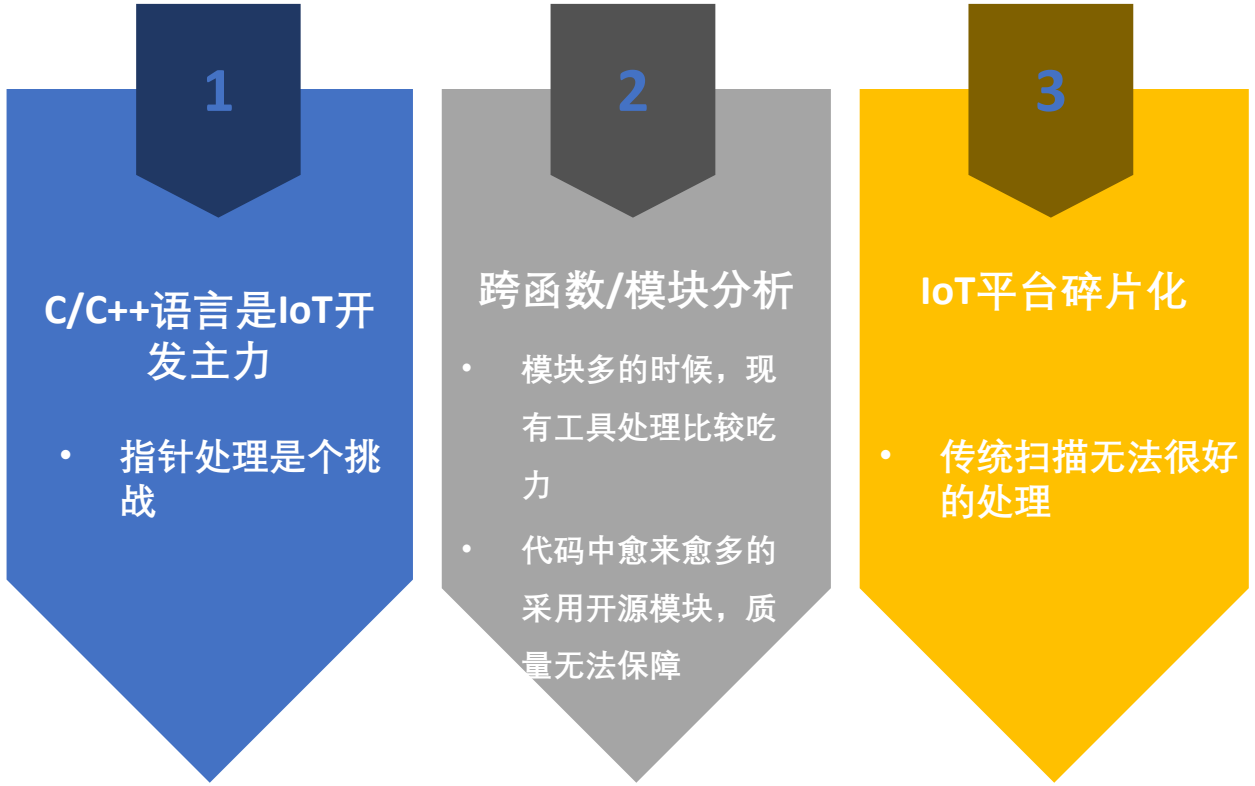
CONTENTS

01 IoT源码分析的挑战

02 基于IR码的源码分析如何应对

03 优势

IoT源码漏洞分析的背景和挑战



IoT源码分析的背景和挑战-平台碎片化

ARM968E-S
- MXCHIP MOC108
- MXCHIP MK3060

ARM Cortex-M0+
- ✓ ST Micro STM32 L0
  - MXCHIP EML3047
  - NUCLEO-L073RZ
  - MXCHIP STM32L071
- ✓ NXP Kinetis® L Series
  - FRDM-KL27Z(MKL27Z64VLH4)
  - MXCHIP ST BlueNRG-1

ARM Cortex-M3
- ✓ MXCHIP MX1101
  - MK1101
- ✓ Opulinks
  - OPL1000

ARM Cortex-M4
- ✓ ST Micro STM32 F4/L4
  - B-L475E-IOT01A
  - NUCLEO-L432KC
  - 32L496GDISCOVERY
  - MXCHIP STM32L412
- ✓ NXP LPC5410x
- ✓ LPCXpresso54102
- ✓ GigaDevice GD32 F4
- ✓ Eastsoft ES8P508x
- ✓ MXCHIP
  - MK3166
- ✓ REALTEK IoT Low-Energy SoC
  - RTL8710BN
  - RTL8710BX
- ✓ South Silicon Valley
  - SSV6266P
- ✓ RDA
  - RDA5981
- ✓ Cypress
  - PSoC 6
- ✓ XradioTech
  - XR871
- ✓ Actions Technology
  - ATS3503
- ✓ Beken
  - BK3435
  - BK7231
- ✓ TI
  - CC3220S
- ✓ Nordic
  - nRF52840

ARM Cortex-M7
- ✓ ST Micro STM32 F7
  - STM32F769I-DISCO

Xtensa
- ✓ LX6 (ESP32)
  - ESP32-DevKitC
- ✓ LX106 (ESP8266)
  - ESP8266

AndesCore
- ✓ MVSILICON P20
- ✓ C-SKY
- ✓ CK801
  - CK802
  - CH2201
  - CB2201
- ✓ CK803
- ✓ CK807
- ✓ CK810
- ✓ CK860
- ✓ CK610

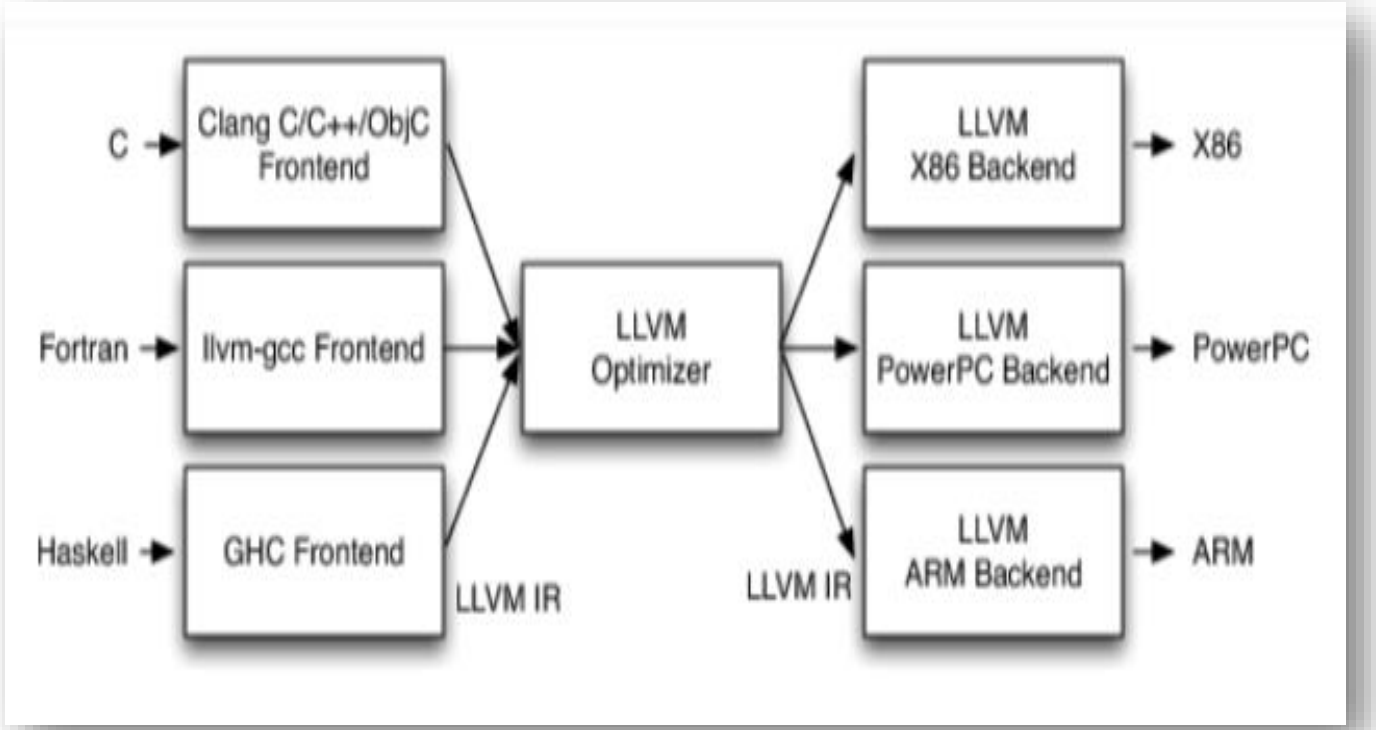
Renesas
- ✓ RL78 Family
  - R7F0C004
- ✓ RX Family
  - RX65N

Linux
- ✓ user mode simulation
  - linuxhost

Ali OS
50 board, 43 MCU！

AliOS部分需要支持的MCU和平台

以LLVM为例理解一种编译过程



LLVM特点

- 不同的前端后端使用统一的中间代码LLVM Intermediate Representation (LLVM IR)
- 如果需要支持一种新的编程语言，那么只需要实现一个新的前端
- 如果需要支持一种新的硬件设备，那么只需要实现一个新的后端
- 优化阶段是一个通用的阶段，它针对的是统一的LLVM IR，不论是支持新的编程语言，还是支持新的硬件设备，都不需要对优化阶段做修改

从一个漏洞看传统静态分析的弱点：CVE-2019-5786


```
Merge M72: FileReader: Make a copy of the ArrayBuffer when returning partial results.

This is to avoid accidentally ending up with multiple references to the
same underlying ArrayBuffer. The extra performance overhead of this is
minimal as usage of partial results is very rare anyway (as can be seen
on https://www.chromestatus.com/metrics/feature/timeline/popularity/2158).
```

- It’s an “Use after Free” bug in Chrome, fixed on March ’19
- It states: “having multiple references to the same underlying ArrayBuffer is a bad thing”
- 谷歌威胁分析团队（Google's Threat Analysis Group）发现了一例野外攻击中的Chrome远程代码执行漏洞:CVE-2019-5786。攻击者利用该漏洞配合一个win32k.sys的内核提权（CVE-2019-0808）可以在win7上穿越Chrome沙箱

从一个漏洞看传统静态分析的弱点：CVE-2019-5786

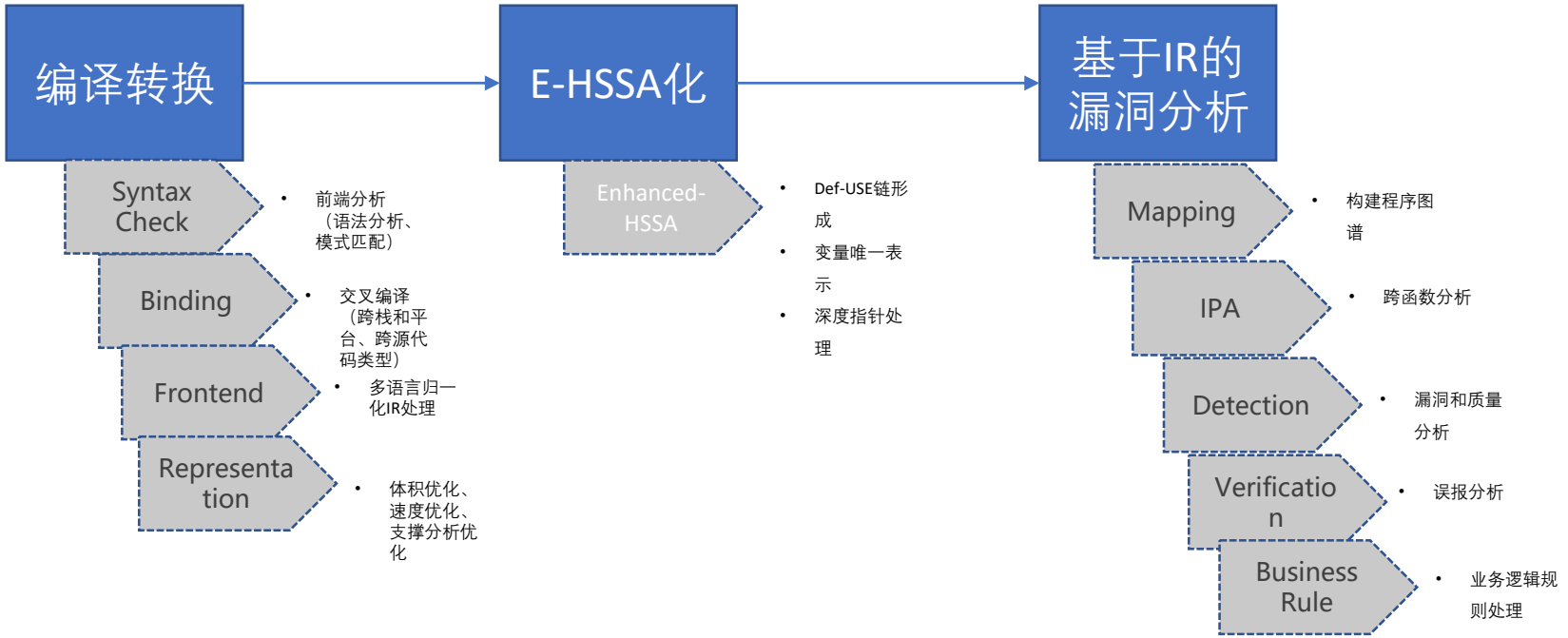
```
@@ -143,14 +143,16 @@
   if (!raw_data_ || error_code_ != FileErrorCode::kOK)
     return nullptr;

-  DOMArrayBuffer* result = DOMArrayBuffer::Create(raw_data_->ToArrayBuffer());
-  if (finished_loading_) {
-    array_buffer_result_ = result;
-    AdjustReportedMemoryUsageToV8(
-        -1 * static_cast<int64_t>(raw_data_->ByteLength()));
-    raw_data_.reset();
+  if (!finished_loading_) {
+    return DOMArrayBuffer::Create(
+        ArrayBuffer::Create(raw_data_->Data(), raw_data_->ByteLength()));
   }
-  return result;
+
+  array_buffer_result_ = DOMArrayBuffer::Create(raw_data_->ToArrayBuffer());
+  AdjustReportedMemoryUsageToV8(-1 *
+                                static_cast<int64_t>(raw_data_->ByteLength()));
+  raw_data_.reset();
+  return array_buffer_result_;
 }
```

- 这个bug被利用的条件是finished_loading设置为true之前，ArrayBufferBuilder :: ToArrayBuffer（）被多次访问，访问方可以短暂获得这个指针的访问和操作权
- ArrayBufferBuilder :: ToArrayBuffer（）是一个三方API实现

1、传统静态代码分析很难理解和发现涉及指针操作和内存的问题

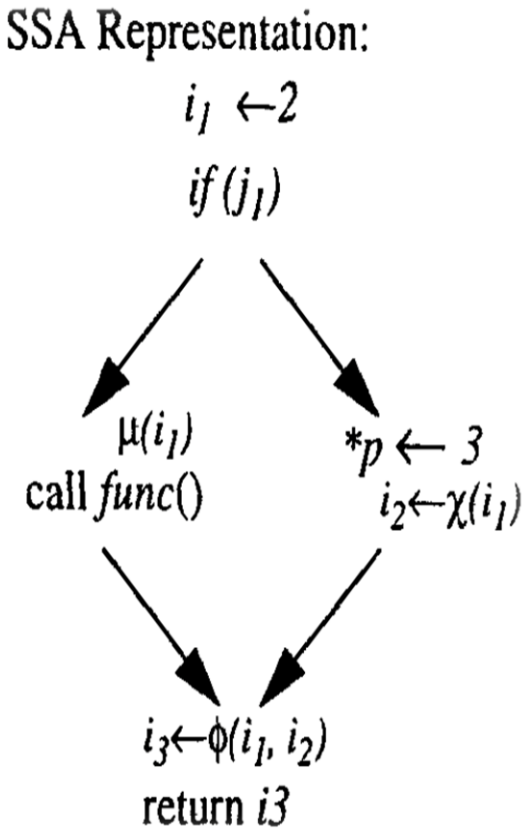
2、传统静态代码分析很难处理跨函数、跨模块的问题。当今的软件工程超过70%采用三方和开源组件、模块，这个问题被进一步放大

基于IR码的漏洞分析解决问题的思路

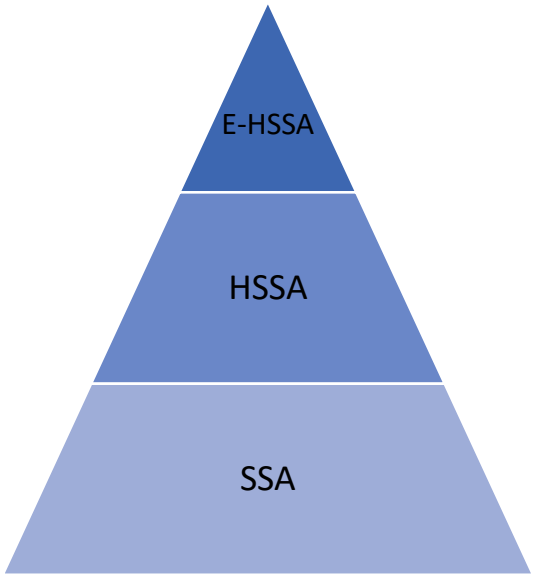


- 基于IR码的源码分析旨在在程序的近执行态基础上构建对象全生命周期视图
- E-HSSA被用于IR的优化，优化目标是形成Def-Use链，E-HSSA加强

Enhanced-HSSA的原理



- SSA是对象Def-Use关系的稀疏表示
- 它集成了对象的控制流和值流
- 原始SSA已经演变为HSSA，即内存SSA，以解决表示别名和编译器优化的间接内存操作的问题
- μ适用于May-Use， χ适用于May-Def
- Phi根据程序的实际运行选择正确的版本
- 我们的Enhanced-HSSA在HSSA基础上构建memory



IR漏洞分析原理

基于E-HSSA形成的Def-Use链去进行分析的关键点

（成份）

- **调用语句、参数和返回值**
- **ISTORE声明**
- **ILOAD表示**
- **程序的入口点暴露使用**
- **退出程序**

- E-HSSA的Memory Model构建实现了对象敏感性
- 基于Def-Use基础的程序图谱映射分析支撑了控制流敏感性
- 基于优化后的IR分析确保了上下文敏感性

示例-UAF（Use After Free）的分析效果

```c
#include <stdio.h>
#include <stdlib.h>

int g = 2;

void my_free(void *p) {
  if (p != NULL)
    free(p);  /* free p */
}

int main() {
  int i, j, *p, *q;
  p = malloc(10 * sizeof(int));
  if (p == NULL)
    return 1;
  for (i=0; i < 10; ++i)
    p[i] = i;
  q = p;
  my_free(p);  /* p is freed */
  j = 0;
  for (i=0; i < 10; ++i)
    j += q[i]; /* read of p[i] is illegal
                  Use-After-Free here */
  return j;
}

```

/home/stone/Documents/testcase/basic/uaf.c

Line: 22 VUL - UAF, Memory referenced through variable in function main has been freed but also referenced afterwards

Line: 22

```c
for (i=0; i < 10; ++i)
  j += q[i]; /* read of p[i] is illegal
                Use-After-Free here */
```

Line: 19

```c
q = p;
my_free(p);  /* p is freed */
j = 0;
```

Line: 8

```c
if (p != NULL)
  free(p);  /* free p */
}
```

Line: 19

```c
q = p;
my_free(p);  /* p is freed */
j = 0;
```

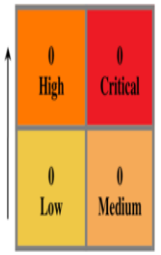
Executive Summary

This workbook is intended to provide all necessary details and information for a developer to understand and remediate the different issues discovered during the basic_1 project audit. The information contained in this workbook is targeted at project managers and developers.

This section provides an overview of the issues uncovered during analysis.

Project Name:	basic_1
Project Version:	
SCA:	Results Present
WebInspect:	Results Not Present
SecurityScope:	Results Not Present
Other:	Results Not Present

Issues by Priority



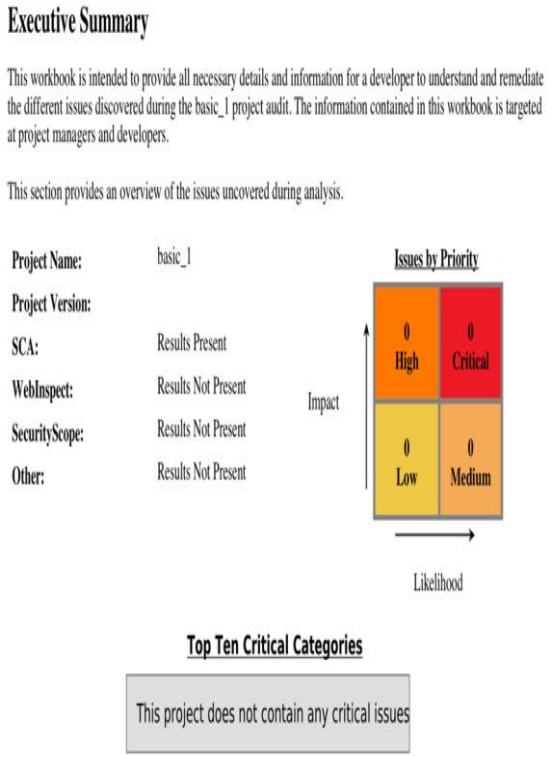
Top Ten Critical Categories

This project does not contain any critical issues

UAF:当堆内存被释放后又重新申请内存时,新申请的内存与刚释放的内存有重合的部分，被释放的内存虽然已被释放掉,但指向该内存的指针还存在,如果操作该指针就可能修改掉新申请内存数据．如当在堆中创建一个对象p指向它,该对象释放后,又重新申请堆内存,而且这2次内存有重合的部分,此时新的内存如果可控,且又引用了该指针(如调用虚函数),即可能执行任意代码

示例-DBF（Double Free）的分析效果

```c
#include <stdio.h>
#include <stdlib.h>


int func_1(void *p) {
  if (p != NULL) {
    free(p);   /* free p */
  }
}

int func_2(void *p) {
  if (p != NULL) {
    free(p);  /* free p */
  }
}

int main() {
  int i, *p, *q;
  p = malloc(10 * sizeof(int));
  if (p == NULL)
    return 1;
  for (i=0; i < 10; ++i)
    p[i] = i;
  q = p;
  func_1(p);  /* free p the first time */
  func_2(q);  /* free p the second time */
  return 0;
}
```



DBF:触发内存重复释放，可能会导致系统重启或执行任意代码

Benefit In IoT

- 基于基本的交叉编译处理，能良好应对IoT碎片化平台的场景分析
- 经过E-HSSA化的IR码，在Def-Use链基础上实现了上下文、数据流和对象敏感性，能够非常好的处理跨函数、跨模块的分析。在此基础上的分析不再受限不同的软件供应链生态
- 基于E-HSSA的分析，良好处理涉及指针分析的问题（跨多层深度），尤为擅长对C和C++语言的分析

THANK YOU

梆 梆 安 全 保 护 智
能 生 活