IoT时代LLVM编译器防护的艺术

刘柏江　　几维安全创始人兼CTO

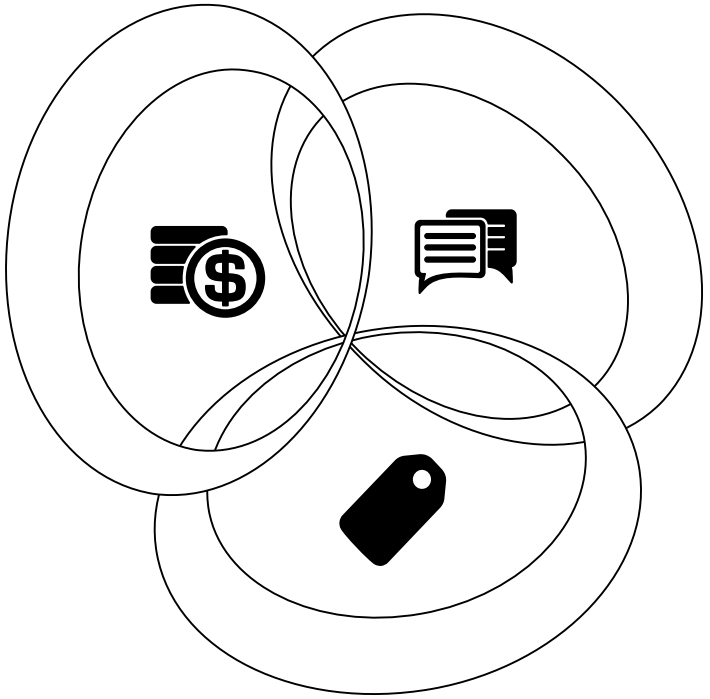
ISC 互联网安全大会 中国 · 北京
Internet Security Conference 2018 Beijing · China

目录

万物互联，安全先行

物理安全

防止丢失或者被盗



系统安全

防止底层漏洞被恶意利用

业务安全

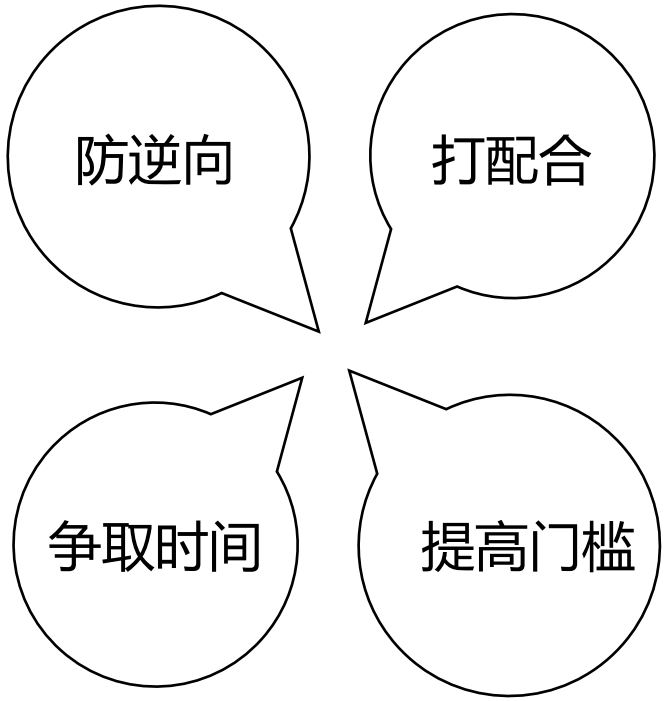
防止用户隐私数据泄漏

ZERO TRUST SECURITY

策略安全

万物互联，代码安全先行

防止核心算法被重构



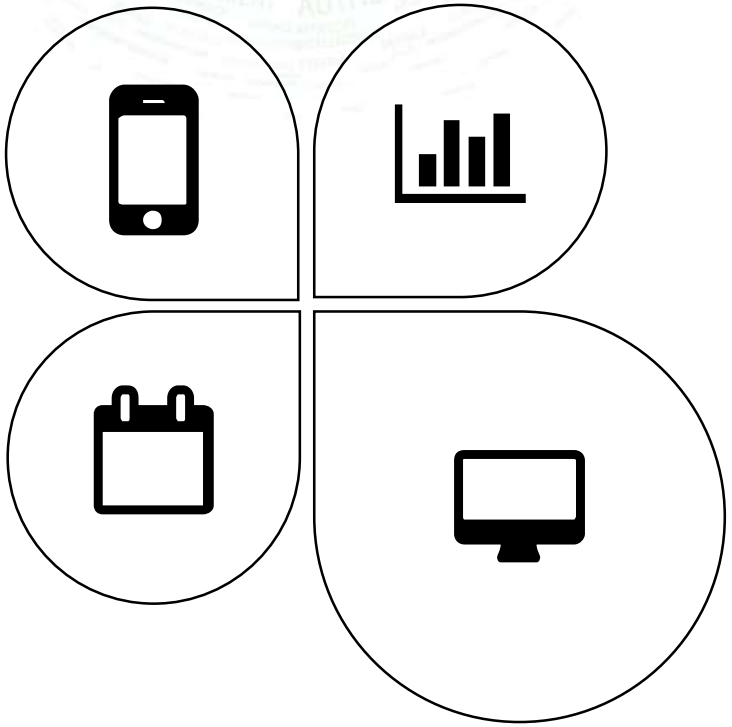
提升策略安全的强度

加大破解难度，延长破解时间，为运营争取更多的有利窗口期

提高破解成本，将逆向菜鸟拒之门外

ZERO TRUST SECURITY

物联网时代即将开启



Android Things

让您可以为各种消费者，零售和工业应用程序构建智能互联设备。

AliOS Things

面向IoT领域的轻量级物联网嵌入式操作系统，可广泛应用在智能家居、智慧城市、新出行等领域。

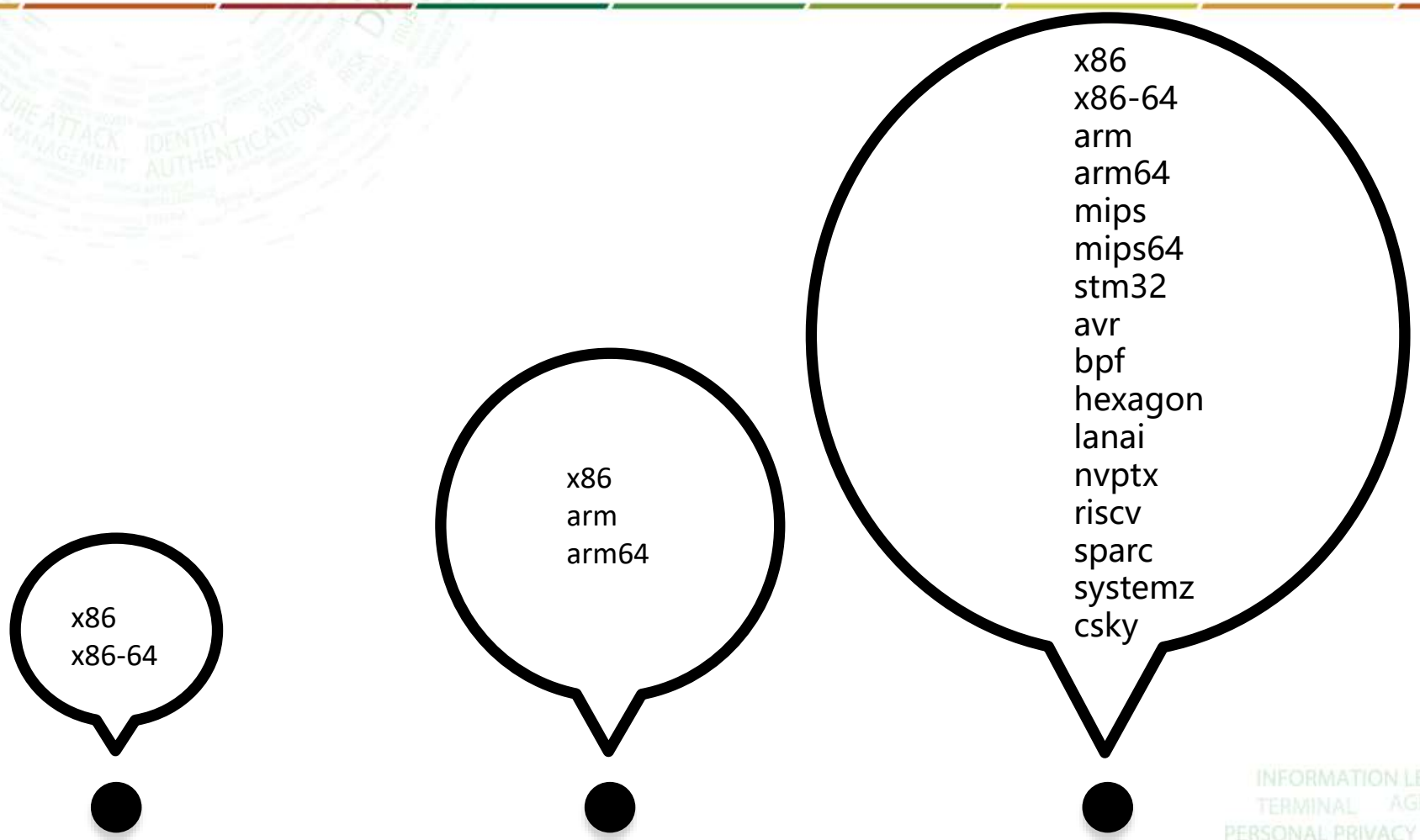
DuerOS

可以广泛支持手机、电视、音箱、汽车、机器人等多种硬件设备

IoT.MI

小米IoT开发者平台面向智能家居、智能家电、健康可穿戴、出行车载等领域

芯片体系越来越多



互联网
Windows / MacOS / Linux

移动互联网
iOS / Android

物联网
Android / AliOS Things

运行内存越来越少

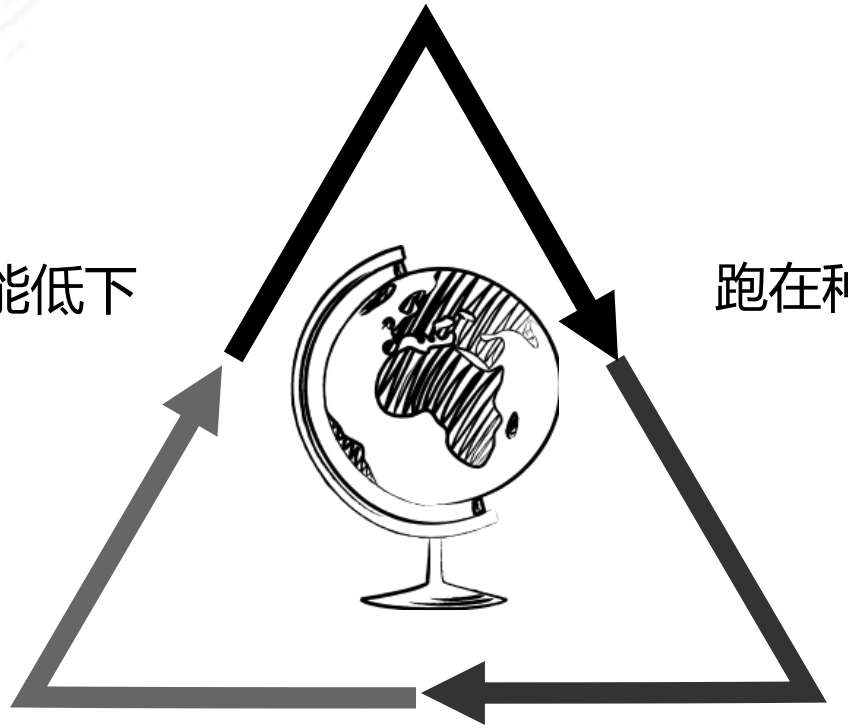
互联网
Windows / MacOS / Linux

移动互联网
iOS / Android

物联网
Android / AliOS Things

物联网操作系统运行环境

受限于功耗硬件性能低下



跑在种类繁多的芯片架构上

运行在内存一般偏小的环境

概念科普

黑盒代码加密

```
01000101001000110101010010101010
00101001000110101010010101010001
01001000110101010010101010001010
01000110101010010101010001010010
00110101010010101010001010010001
10101010010101010001010010001101
01010010101010001010010001101010
10010101010001010010001101010100
10101010001010010001101010100101
01010001010101000101010100010101
```

处理的对象是最终的软件执行体，比如Windows的EXE、Android的SO以及DEX

白盒代码加密

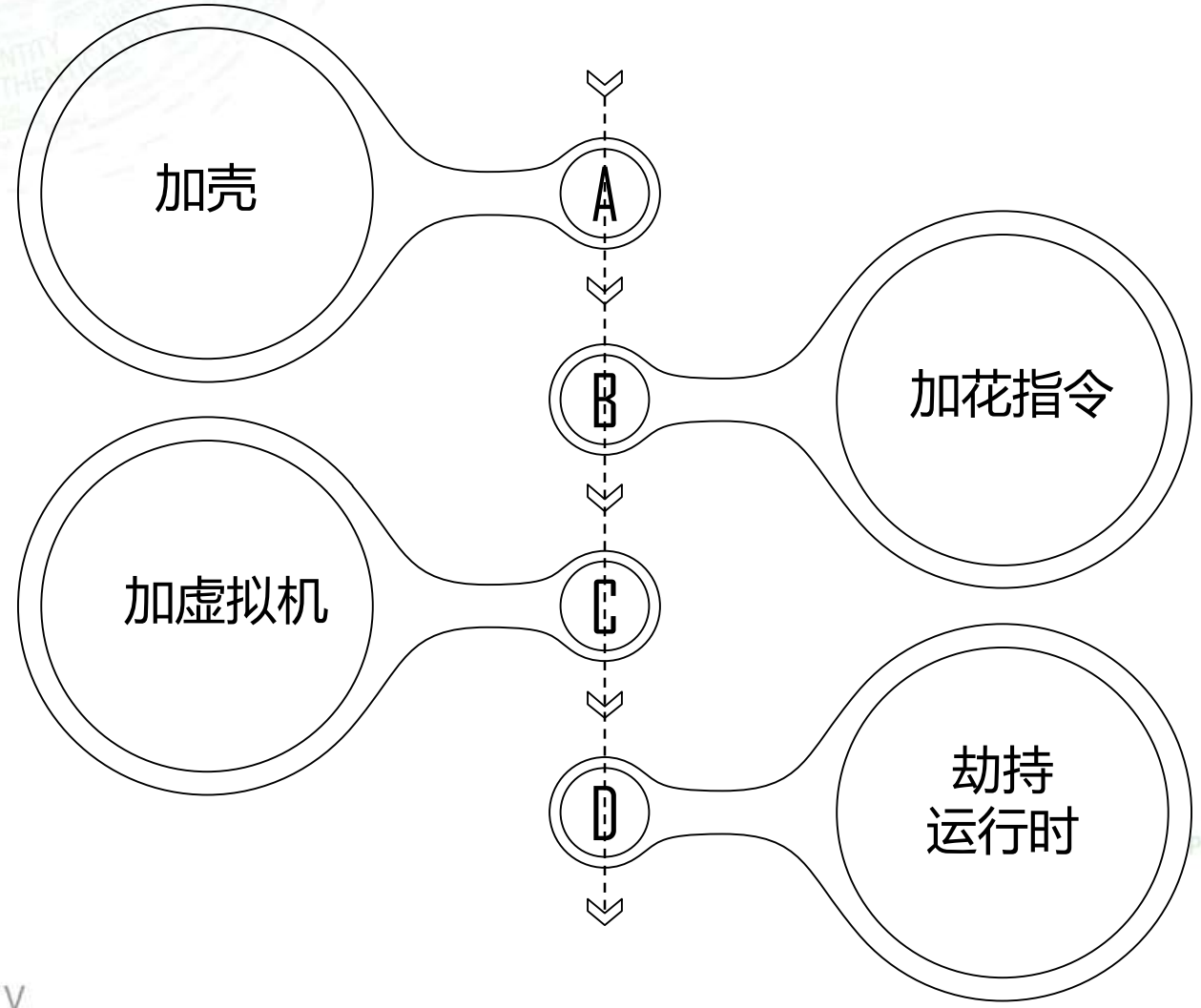
```
int binary_search(int val[] , int num , int value)
{
        int start = 0;
        int end = num - 1;
        int mid = (start + end) / 2;

        while (val[mid] != value && start < end) {
                if (val[mid] > value) {
                        end = mid - 1;
                }
                else if (val[mid] < value) {
                        start = mid + 1;
                }
                mid = ( start + end )/2;
        }
        if (val[mid] == value) {
                return mid;
        }
        else {
                return -1;
        }
}
```

处理的对象是源代码，比如C/C++/Objective-C / Swift这类语言的源代码文件

黑盒代码加密的应用

比如适用于Windows、Linux、Android的UPX壳

比如Windows非常著名的VMProtect



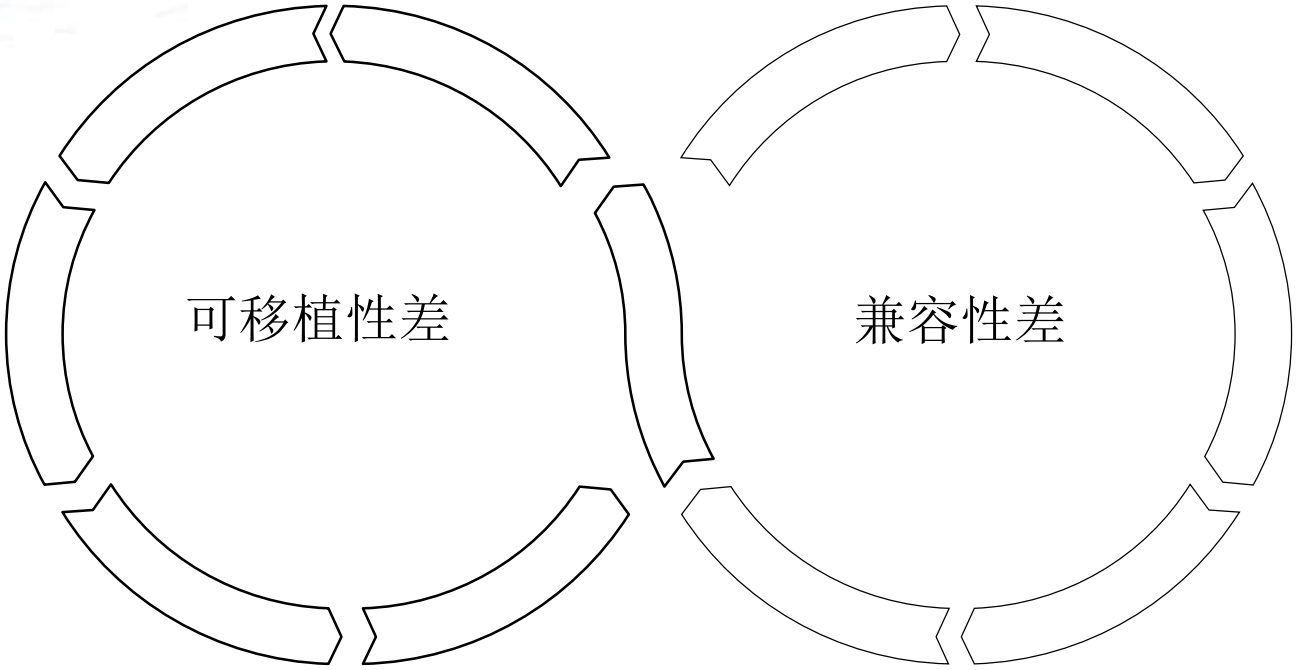
比如利用x86指令集的可变长特性增加误导反汇编程序的垃圾指令

比如Android的DEX整体加解密、类抽取

黑盒代码加密的局限

很难对多端且同源的代码做一致性的保护

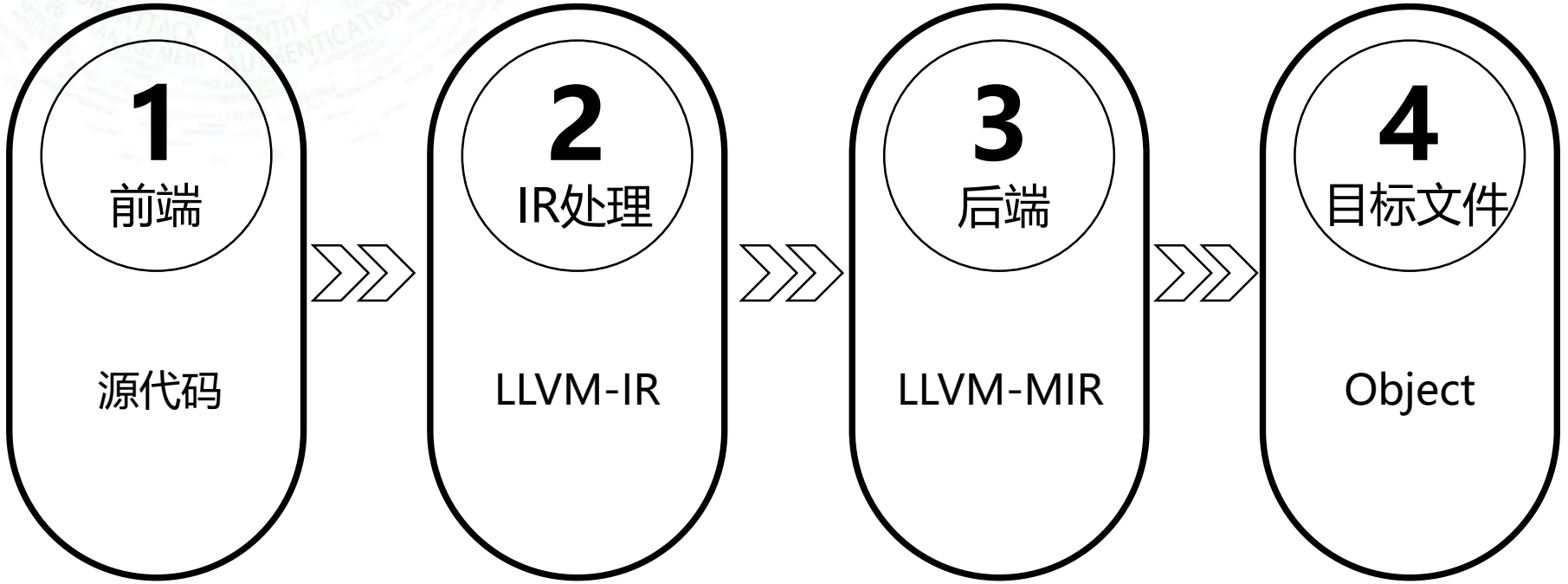
对于像Android这类高碎片化的平台，干预运行时意味着兼容性极差



芯片架构不兼容、内存需求显著增加，很难适应新的像IoT这样的平台

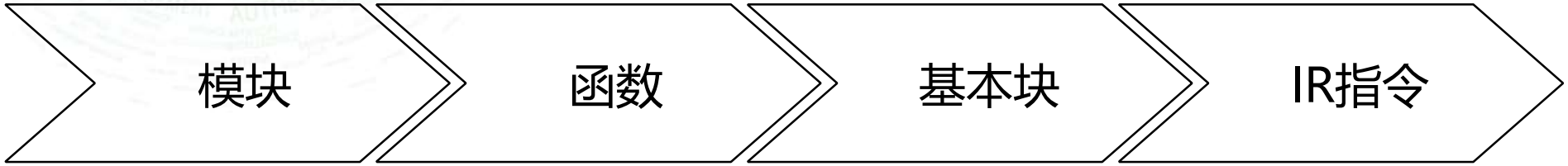
对于像iOS这类完全封闭的平台，干预运行时意味着方案没法工作

LLVM编译器登场



LLVM是模块化、可复用的编译器工具链集合，最初是伊利诺伊大学的一个研究项目，其目标是提供一种现代的，基于SSA的编译策略，能够支持任意编程语言的静态和动态编译。

LLVM-IR

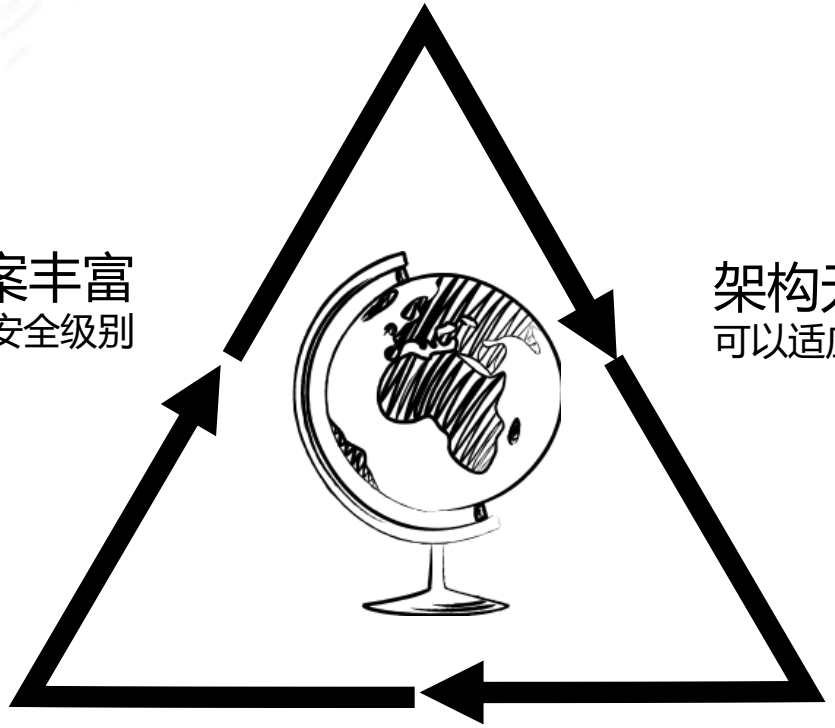


模块	函数	基本块	IR指令
架构信息 全局数据 函数申明 函数实现 编译元数据	函数申明 变量申明 代码块	IR指令序列	算术指令 逻辑指令 控制指令 ……

LLVM提供了完整的IR文件操作API，可以对IR文件的模块、函数、基本块、IR指令做任意修改。

挖掘LLVM-IR的潜能

方案丰富
可以满足任意要求的安全级别

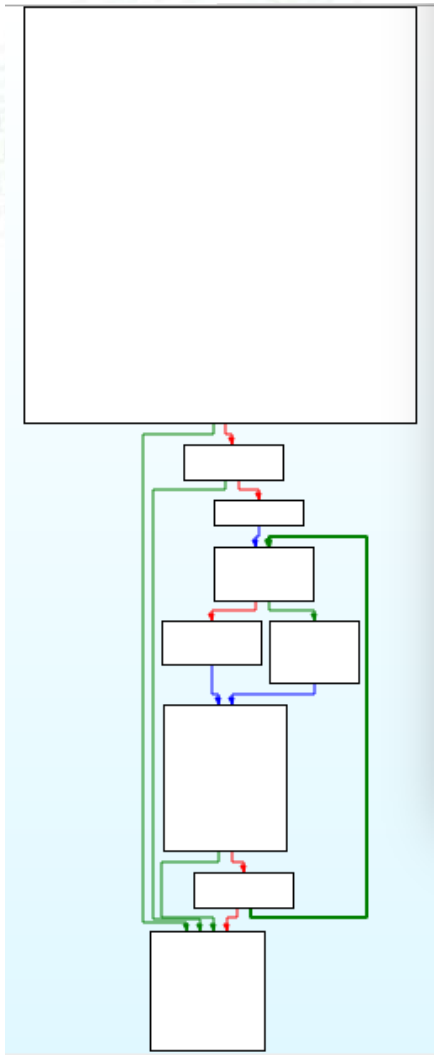


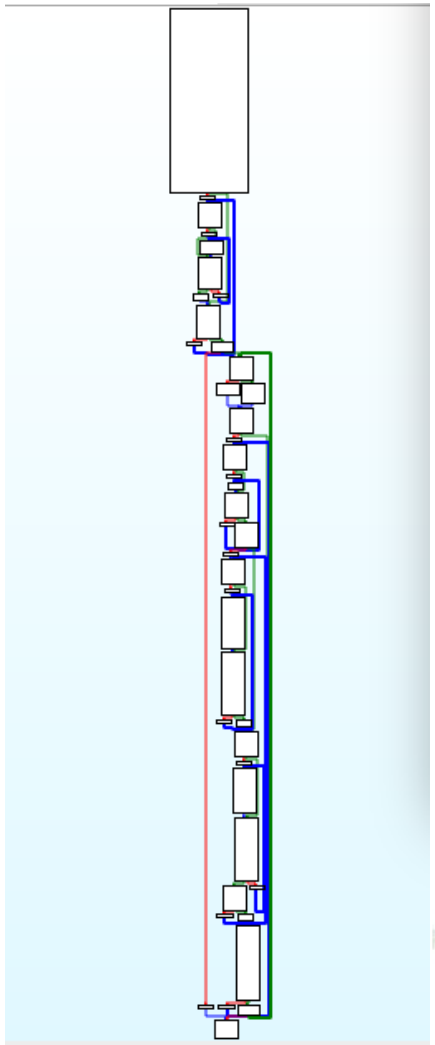
架构无关
可以适应任意芯片架构

函数粒度
可以适应低内存运行环境

初级防护

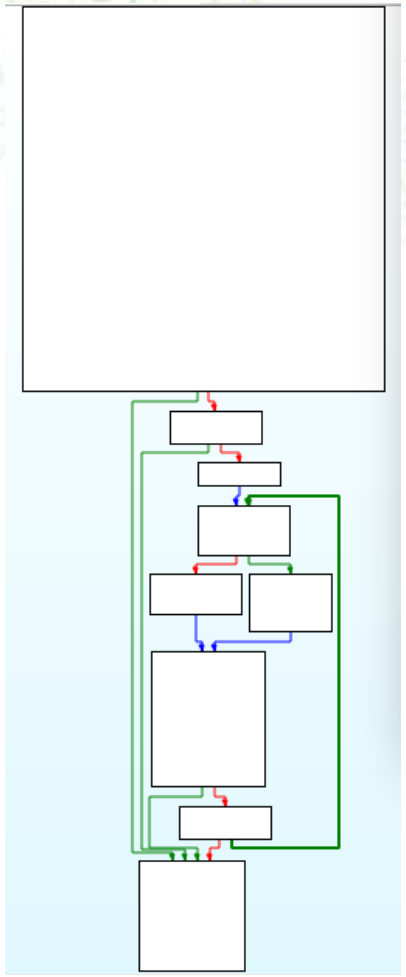
原始流程图



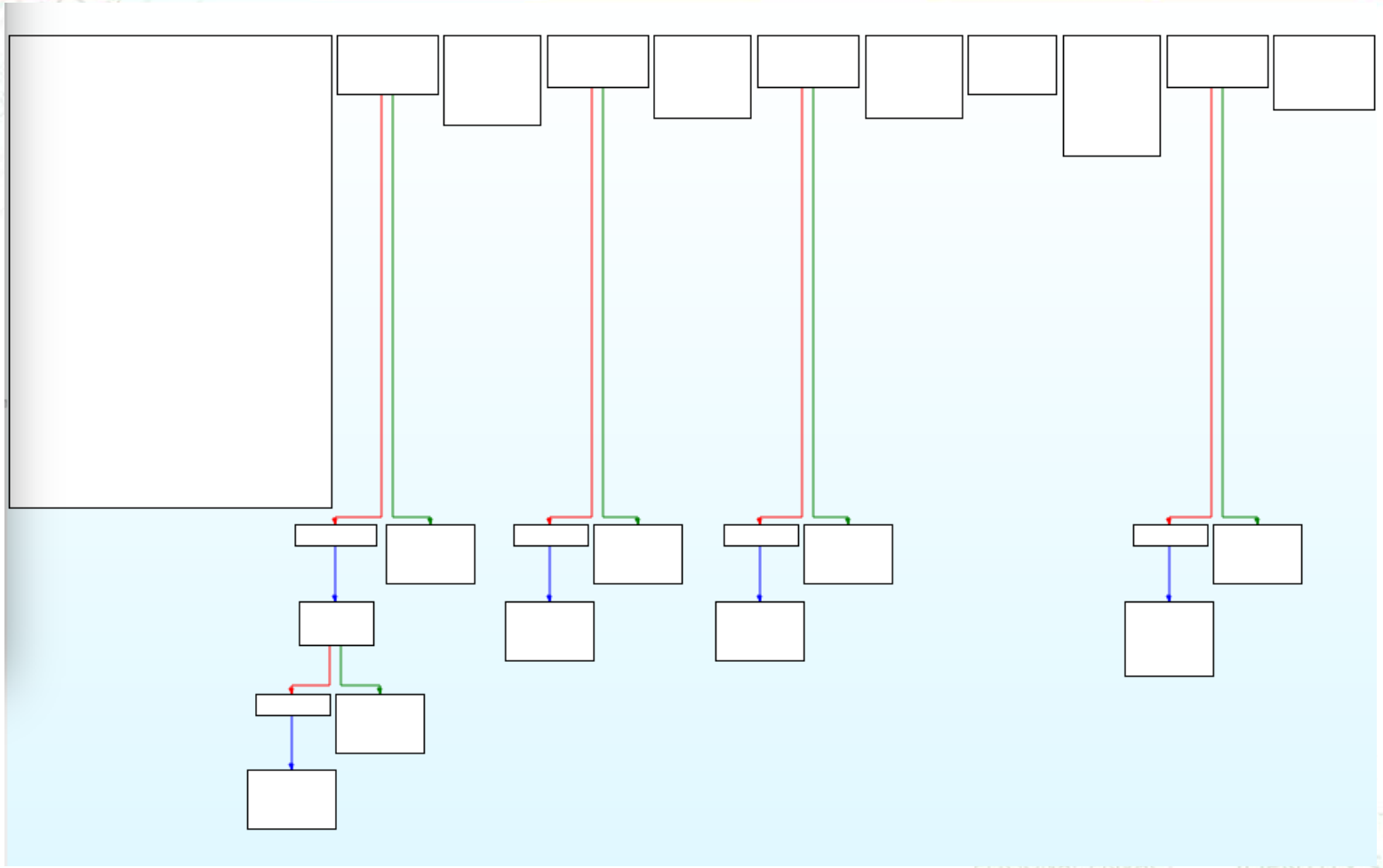


混淆流程图

高级防护



原始流程图



块调度流程图

代码虚拟化KIWIVM

虚拟CPU执行

函数保护粒度

全平台全架构

100%兼容性

KiwiVM代码虚拟化编译器基于LLVM编译器中间层实现，通过设计独有保密的虚拟CPU指令，将原始CPU指令进行加密转换为只能由KiwiVM解释执行的虚拟指令，能够完全隐藏函数代码逻辑，让代码无法被逆向工程。

KIWIVM转换过程

KiwiVM的中心思想是利用LLVM-BC编码成自定义虚拟CPU的指令集和元数据，包括指令集数据、重定位数据、函数调用签名数据等。

KIWIVM转换样例

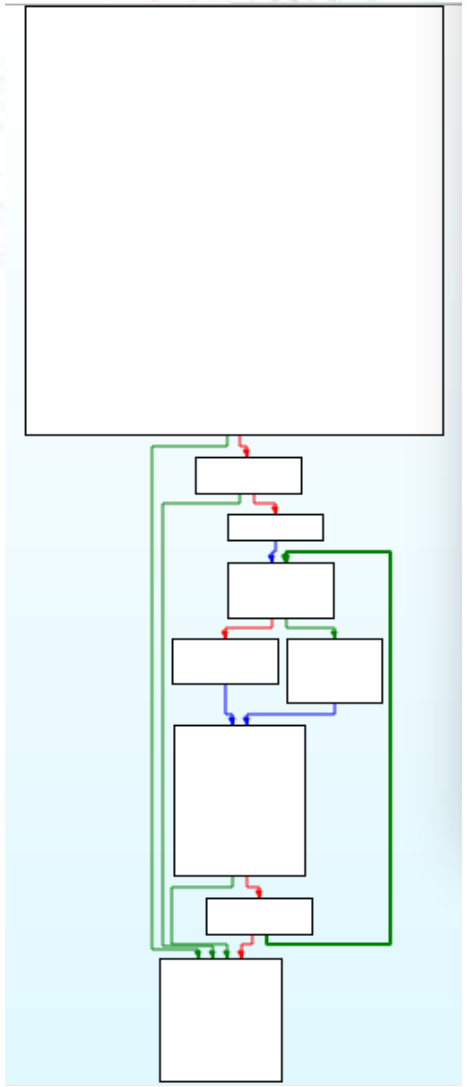
```c
extern int puts(const char *);

int binary_search(int val[] , int num , int value)
{
    int start = 0;
    int end = num - 1;
    int mid = (start + end) / 2;

    puts("KiwiVM Demo");

    while (val[mid] != value && start < end) {
        if (val[mid] > value) {
            end = mid - 1;
        }
        else if (val[mid] < value) {
            start = mid + 1;
        }
        mid = ( start + end )/2;
    }
    if (val[mid] == value) {
        return mid;
    }
    else {
        return -1;
    }
}
```

```c
//原始函数
i32 KVMEXPORT binary_search(void * p1, i32 p2, i32 p3) {
    const void *__kvm_argv__[] = {&p1, &p2, &p3};
    KVMInterpContext kvm = {
        __KVM_RELOCS__, //重定位数据
        __KVM_SIGNS__,  //函数签名
        __KVM_MODULE__, //指令编码数据
        __kvm_init_gv__, __kvm_api_bridge__, 209, 3, argv
    };
    //虚拟机入口
    KVMResult __kvm_result__ = *kiwisec_vm_interpreter(&kvm);
    return *(i32 *)&__kvm_result__;
}

//函数签名
KVMHIDDEN const unsigned char __KVM_SIGNS__[] = {
 0x01, 0x02, (unsigned char)sizeof(i32), (unsigned char)sizeof(void *),
};

//重定位数据
KVMHIDDEN const KVMRelocation __KVM_RELOCS__[] = {
 { 9, {.value = 0x7fa73e000406ll} },
 { 2, {.value = 0xc40000005ll} },
 { 4, {.ptr = (void *)&KVMputs} },

};

//指令编码数据
KVMHIDDEN unsigned char __KVM_MODULE__[] = {
 0x69, 0x02, 0x00, 0x00, 0x10, 0x03, 0x00, 0x00, 0xE4, 0x60, 0xE4, 0xE3,
 0x49, 0x36, 0x34, 0x00, 0x28, 0x03, 0x00, 0x00, 0xF1, 0xCB, 0x3E, 0x5B,
 ......
};
```

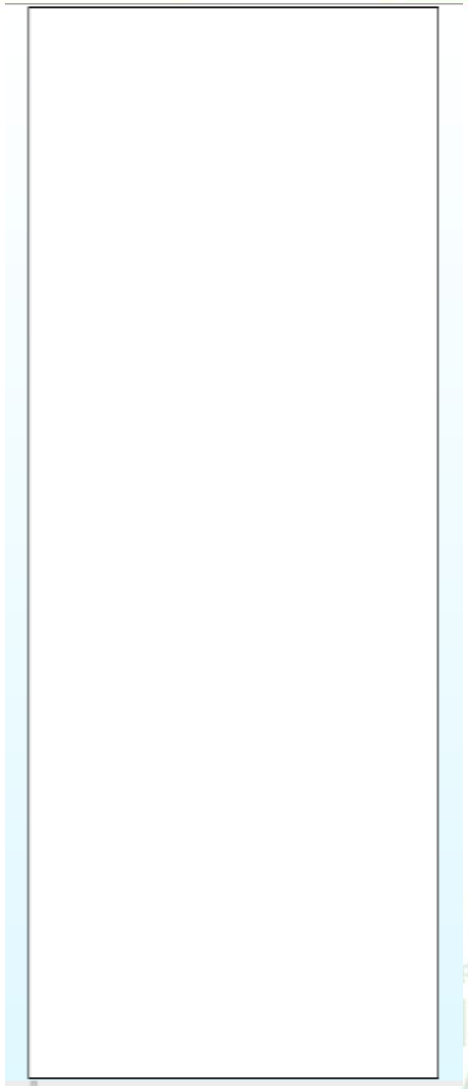
```asm
_binary_search proc near

var_68= byte ptr -68h
var_30= byte ptr -30h
var_10= byte ptr -10h
var_8= byte ptr -8
var_4= byte ptr -4

push    rbp
mov     rbp, rsp
sub     rsp, 70h
lea     rax, [rbp+var_10]
mov     [rax], rdi
lea     rcx, [rbp+var_8]
mov     [rcx], esi
lea     rsi, [rbp+var_4]
mov     [rsi], edx
lea     rdx, [rbp+var_30]
mov     [rdx], rax
mov     [rdx+8], rcx
mov     [rdx+10h], rsi
lea     rax, ____KVM_RELOCS___
lea     rdi, [rbp+var_68]
mov     [rdi], rax
lea     rax, ____KVM_MODULE___
movq    xmm0, rax
lea     rax, ____KVM_SIGNS___
movq    xmm1, rax
punpcklqdq xmm1, xmm0
movdqu  xmmword ptr [rdi+8], xmm1
lea     rax, ___kvm_init_gv__
mov     [rdi+18h], rax
lea     rax, ___kvm_api_bridge__
mov     [rdi+20h], rax
mov     dword ptr [rdi+28h], 0BDh
mov     dword ptr [rdi+2Ch], 3
mov     [rdi+30h], rdx
call    _kiwisec_vm_interpreter
mov     eax, [rax]
add     rsp, 70h
pop     rbp
retn
_binary_search endp
```

旗舰防护

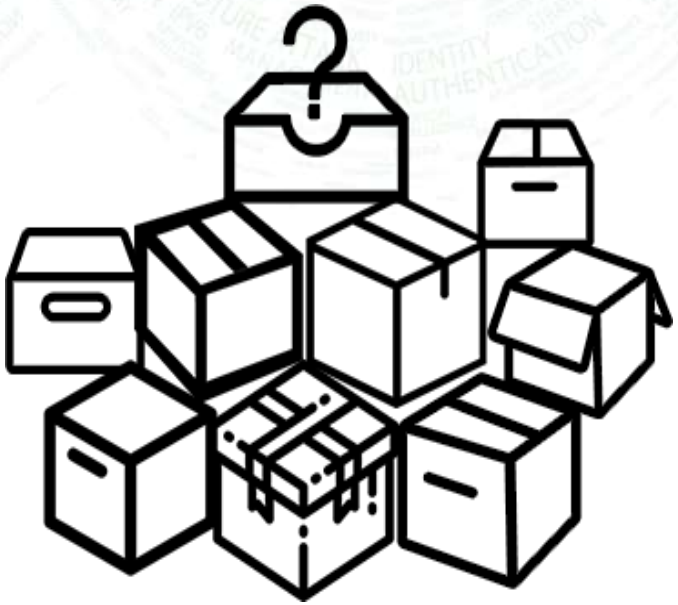
原始流程图



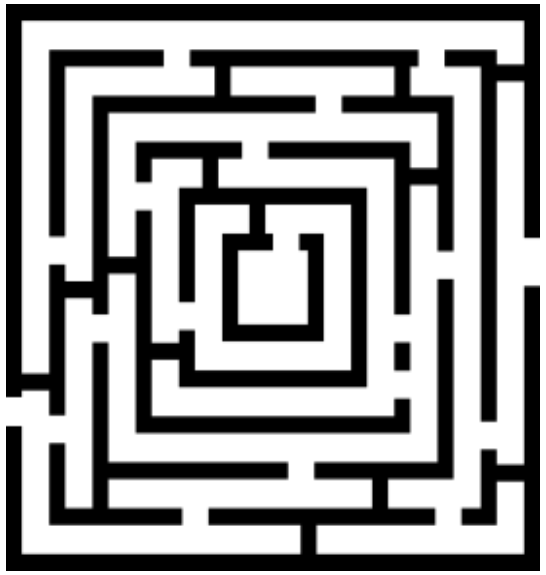
虚拟机流程图



差异对比



混淆



块调度

代码虚拟化

结束之前

几维安全编译器产品简介

	级别	功能	平台
混淆编译器	初级	代码膨胀 乱序执行	iOS、Android、IoT
块调度编译器	高级	逻辑断链 函数调用隐藏	iOS、Android、IoT
虚拟化编译器	旗舰级	逻辑隐藏 虚拟CPU执行	iOS、Android、IoT

ISC 2018
ISC互联网安全大会

360互联网安全中心

谢 谢！

ISC 互联网安全大会 中国 · 北京
Internet Security Conference 2018 Beijing · China