跨平台内存安全测试集的设计与实践

沈思豪 解达 宋威

中国科学院信息工程研究所信息安全国家重点实验室

SKLOIS
信息安全国家重点实验室

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

2022年8月26日

内存安全

内存安全攻击手段

- 缓冲区溢出
- 释放后使用(Use After Free)
- 代码注入攻击
- 代码复用攻击
- 返回导向编程技术(ROP)
- 伪造对象导向编程技术(COOP)
- 数据导向编程技术(JOP)
- ……

越来越复杂、隐蔽

防御手段

- 地址空间随机化
- 栈粉碎保护
- 数据执行保护(DEP)
- 控制流完整性
- GCC VTV/LLVM CFI
- 指针完整性
- 指针标记
- Intel MPX/CET
- ARM PA/MTE
- ……

软件 架构无关

软硬件结合 架构相关

对内存安全测试集的需求

- 可移植性强
  - RISC-V
  - x86 架构硬件安全拓展
  - ARM 架构硬件安全拓展
  - ……
- 可拓展性强
  - 应对日新月异的攻击手段
  - 方便添加新的测例

RIPE测试集

- 一系列缓冲区溢出攻击的集合
- 应用广泛，常用于内存安全相关研究中，作为安全性测例
- 每个测例都受5个变量控制
  - 缓冲区溢出位置
  - 受攻击的代码指针类型
  - 溢出攻击的类型
  - 恶意代码
  - 攻击使用的函数
- 设计思想：枚举变量所有组合

对于缓冲区溢出攻击测试比较完善，
但是对缓冲区溢出以外的漏洞涉及不多

RIPE测试集的问题

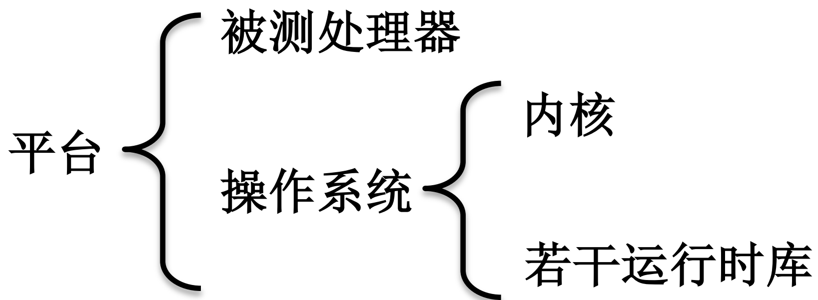
将测试范围从缓冲区溢出扩大到整个内存安全:

- 5个变量 850项测例
- 枚举方法：变量过多 测例数量爆炸
- 需要采用新的思路设计测试集

设计原则

RIPE测试集

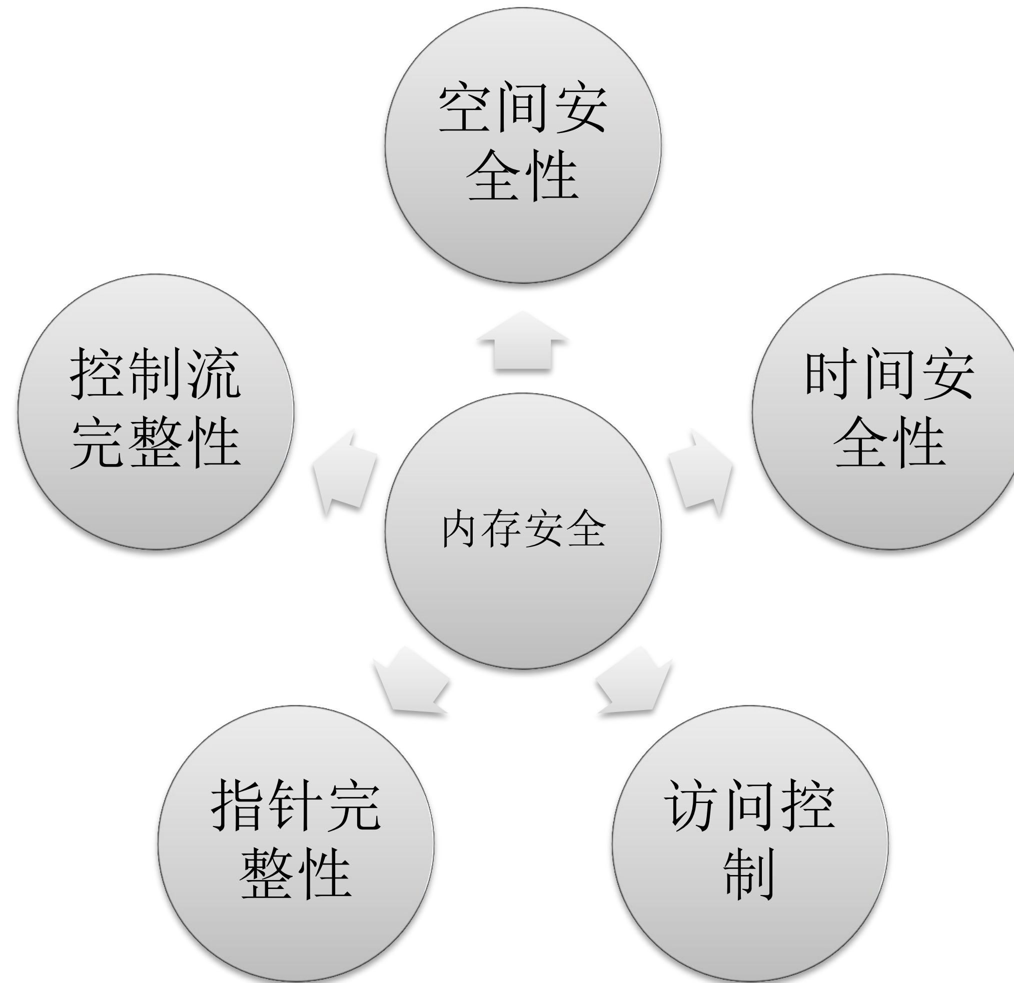
- 利用内存安全漏洞展开完整攻击
- 测试缓冲区溢出保护：
  - 构造完整的缓冲区攻击测例



新设计

- 内存安全作为若干内存安全性质的集合
- 对于利用内存漏洞展开的攻击，我们分析该攻击破坏或利用了哪些内存安全性质，而不关注攻击本身
- 测试缓冲区溢出保护：
  - 缓冲区溢出攻击→内存越界访问性质
  - 针对内存越界访问构造测例
- 以平台为对象进行测试

测例分布



空间安全性

- 空间安全性指内存访问总是落在正确的数据边界和程序的可见域内

覆盖的内容主要包括：

- 访存越界，多个角度
  - 上溢/下溢
  - 访存越界发生位置：栈、堆、全局变量
  - 特殊的访存越界行为：喷射攻击
- 其他可以检测和阻止访存越界的行为
  - 跨只读页、非只读页越界访问
  - 跨数据段、代码段越界访问

时间安全性

- 时间安全性指内存中的数据访问只发生在数据的生命周期之内

覆盖的内容主要包括:

- 释放后使用（UAF）
  - 空悬指针
  - 栈帧复用行为
- 未初始化变量
  - 变量位置：栈、堆、全局变量

访问控制

- 访问控制性质限制了攻击者内存访问的能力

目前包括三项测例:

- 地址空间随机化（ASLR）
- 全局偏移量表（GOT）读取
- 函数体读取

指针完整性

- 指针完整性主要关注：敏感数据指针的安全性.
- 敏感数据指针包括函数指针、虚函数表指针和全局偏移量表

覆盖的内容主要包括：

- 函数指针赋值操作
- 函数指针算术运算操作
- 虚函数表指针读取
- 全局偏移量表修改

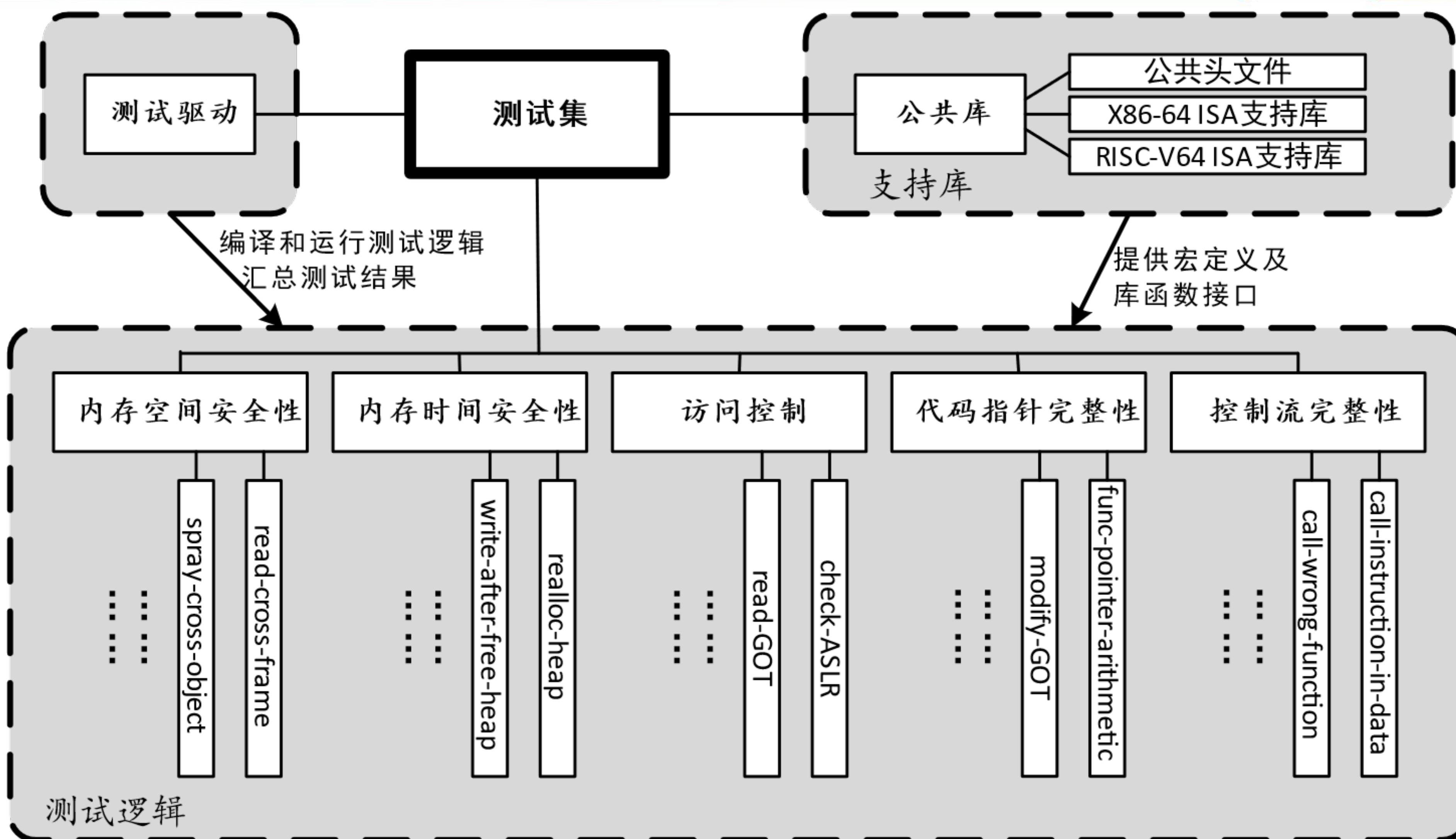
控制流完整性

- 控制流体现了程序动态执行时指令间逻辑上的先后顺序.
- 前向控制流
  - 通过代码指针调用完成的控制流跳转
  - 前向控制流完整性指保护代码指针解引用到合法的地址
- 后向控制流
  - 通过返回地址完成的控制流跳转
  - 后向控制流完整性指保护返回地址不被恶意篡改

典型的测例测试的内容包括:

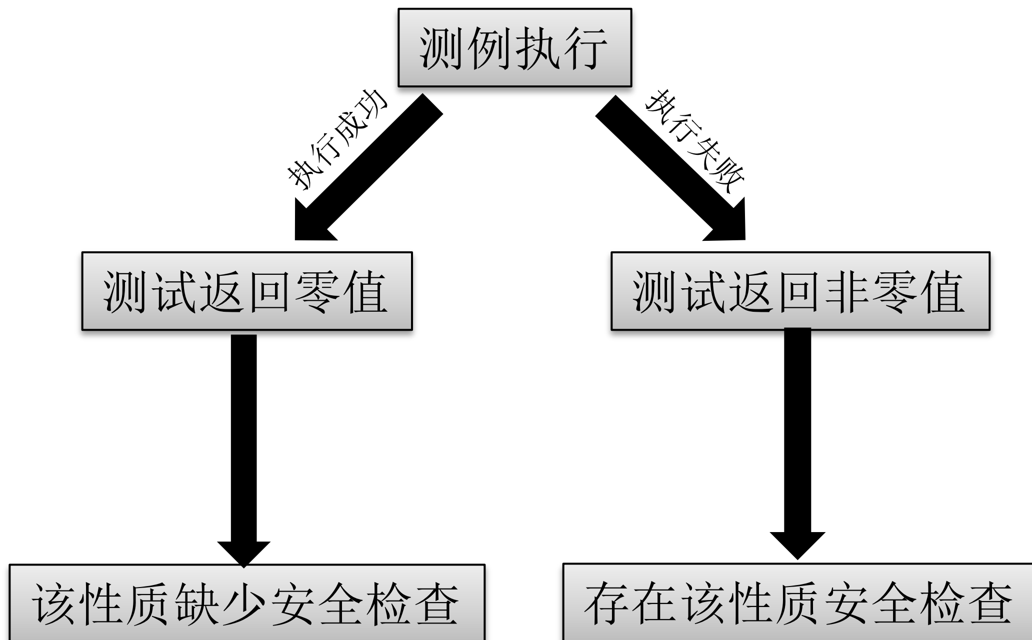
- 前向/后向代码注入
- 前向/后向代码复用
  - 返回导向编程技术(ROP)
  - 伪造对象导向编程技术(COOP)
  - 跳转导向编程技术(JOP)
- 虚函数表(VTable)保护
  - 虚函数表修改
  - 虚函数表替换

测例构造



测试逻辑

- 每个测例为测试特定内存安全性质的C++程序;



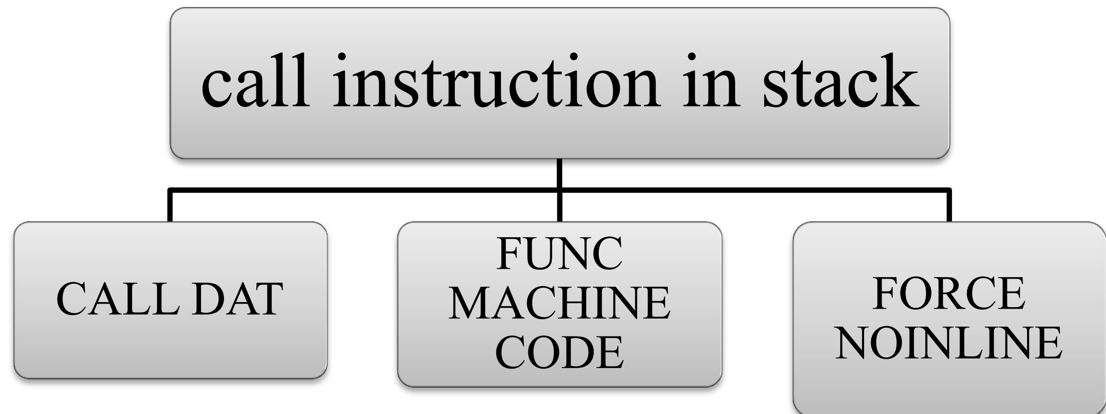

```
#include "include/assembly.hpp"
#include "include/signal.hpp"

int gv = 1;

int FORCE_NOINLINE helper(const unsigned char* m) {
  CALL_DAT(m);
  return gv;
}

int main()
{
 unsigned char m[] = FUNC_MACHINE_CODE;
  … //其他处理
  int rv = helper(m);
  … //其他处理
  exit(rv);
}
```

call-instruction-in-stack的测试逻辑

该测例用来测试将栈上地址作为目标地址的情况下，函数调用是否能够成功执行

支持库

- 恶意行为 → 汇编代码
- 这些汇编代码在平台相关的支持库中




```
#define CALL_DAT(ptr)                                  \
  asm volatile(                                        \
    "jalr ra, %0, 0;"                                  \
    : : "r"(ptr)                                       \
    : "ra"                                             )
```

CALL_DAT宏的RISC-V64架构实现

```
#define CALL_DAT(ptr)                                  \
  asm volatile(                                        \
    "call *%0;"                                        \
    : : "r" (ptr)                                      )
```

CALL_DAT宏的x86架构实现

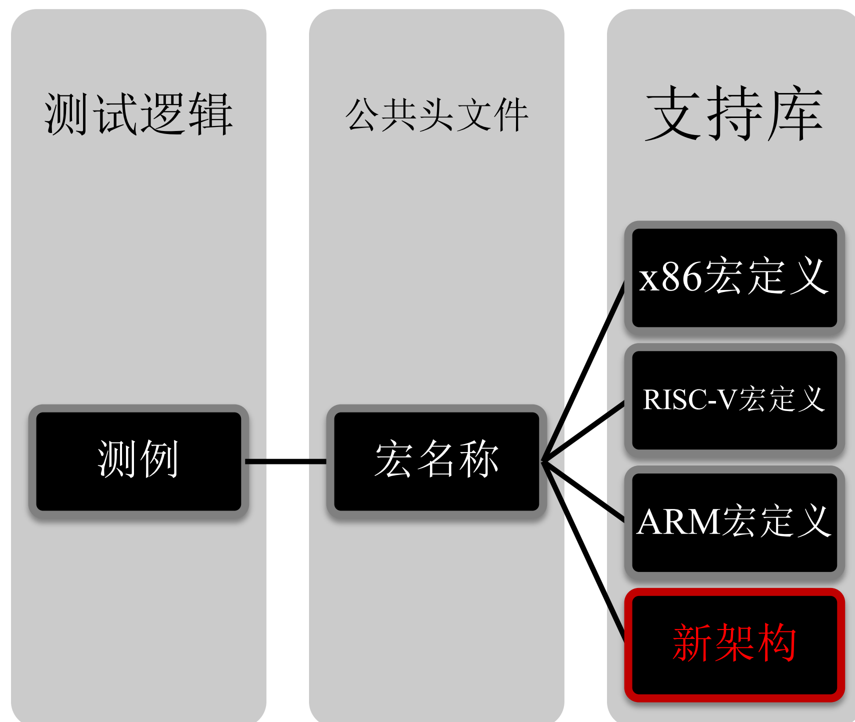
测试驱动

- 测试集成功通过测例数可以量化反映被测平台的内存安全性
  - 通过测例数越多，平台内存安全性越低
- 整个测试集的运行通过测试驱动控制

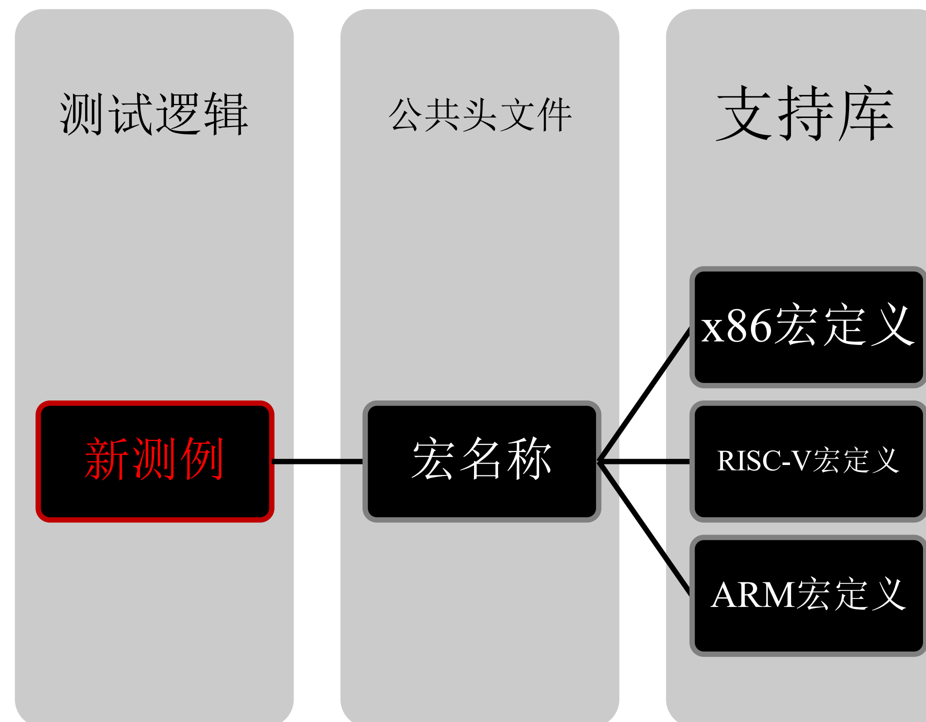
可移植性和可拓展性

可移植性

- 通过测试逻辑与支持库的划分实现.



可拓展性



测试环境

平台	架构	处理器	操作系统	内核版本	编译器	C库
Intel i7-3770	x86-64	Intel i7-3770	Ubuntu 16.04	4.15.0	g++ 5.4.0	GLIBC 2.23
Intel Xeon 8280		Intel Xeon 8280	Ubuntu 18.04	5.4.0	g++ 7.5.0	GLIBC 2.27
HiFive Unleashed	RV64GC	SiFive u540	OpenEmbedded	5.8.2	g++ 10.2.0	GLIBC 2.32
HiFive Unmatched		SiFive u740	OpenEmbedded	5.13.19	g++ 11.2.0	GLIBC 2.28

默认编译选项不同平台通过测例数

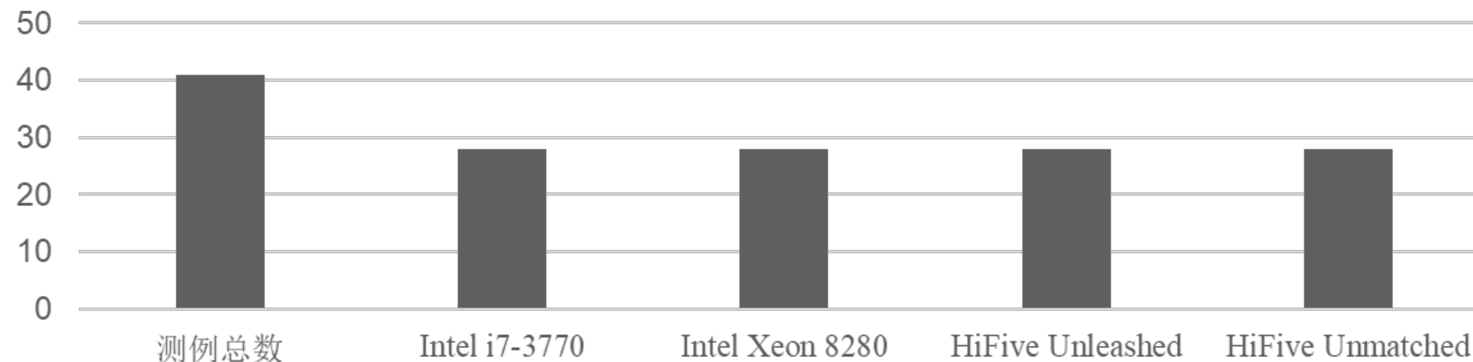
分类	测例总数	Intel i7-3770	Intel Xeon 8280	HiFive Unleashed	HiFive Unmatched
空间安全性	98	98	98	98	98
时间安全性	13	13	13	9	9
访问控制	3	3	2	2	2
指针完整性	5	5	4	5	5
控制流完整性	41	28	28	28	28
总计	160	147	145	142	142

访问控制

测例	Intel i7-3770	Intel Xeon 8280	HiFive Unleashed	HiFive Unmatched
地址空间随机化(ASLR)	成功执行	执行失败	执行失败	执行失败
全局偏移量表(GOT)读取	成功执行	成功执行	成功执行	成功执行
函数体读取	成功执行	成功执行	成功执行	成功执行

控制流完整性

控制流方向	细分攻击手段	Intel i7-3770	Intel Xeon 8280	HiFive Unleashed	HiFive Unmatched	失败原因
后向	代码注入	执行失败	执行失败	执行失败	执行失败	DEP
	其他	成功执行	成功执行	成功执行	成功执行	
前向	代码注入	执行失败	执行失败	执行失败	执行失败	DEP
	其他	成功执行	成功执行	成功执行	成功执行	
虚函数表保护	虚函数表替换	成功执行	成功执行	成功执行	成功执行	
	虚函数表伪造	成功执行	成功执行	成功执行	成功执行	



结论1

- 总的来说, 在各个被测平台上, 由默认配置提供的安全防护并无太大区别.

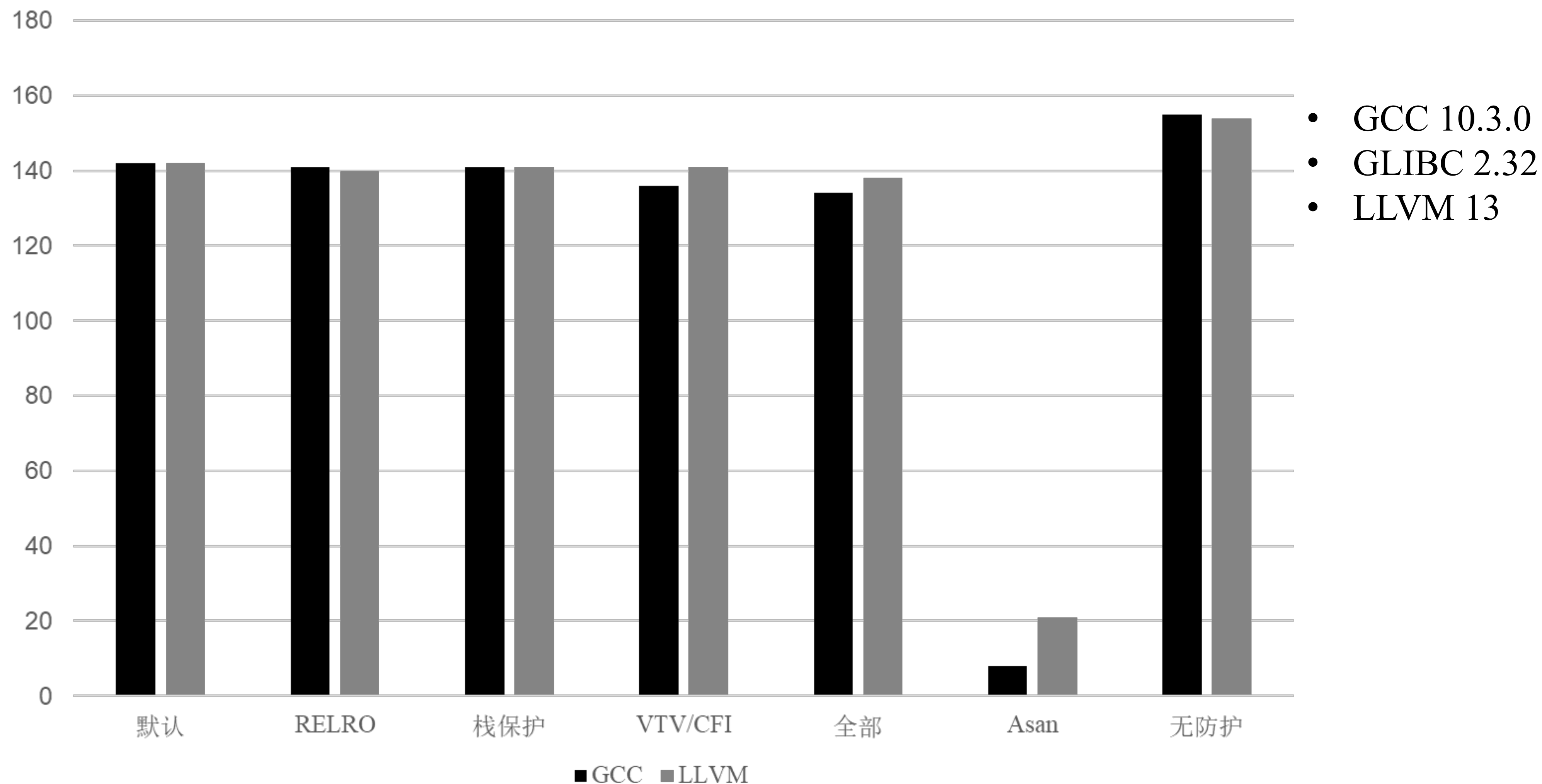
<table>
<tr><th></th><th>Intel i7-3770</th><th>Intel Xeon 8280</th><th>HiFive Unleashed</th><th>HiFive Unmatched</th></tr>
<tr><td>空间安全性</td><td colspan="4">未提供有效防护</td></tr>
<tr><td>时间安全性</td><td colspan="2">未提供有效防护</td><td colspan="2">配套工具链和运行时库提供了安全防护</td></tr>
<tr><td>访问控制</td><td>未开启有效防护</td><td colspan="3">地址空间随机化(ASLR)提供了部分防护</td></tr>
<tr><td>指针完整性</td><td colspan="4" rowspan="2">数据执行保护(DEP)提供了部分防护</td></tr>
<tr><td>控制流完整性</td></tr>
</table>

Intel平台不同编译选项

选项组名称	编译选项	支持的编译器
默认	-O2	GCC 和 LLVM
RELRO	-pie -fPIE -Wl,-z,relro,-z,now	GCC 和 LLVM
栈保护*	-Wstack-protector -fstack-protector-all	GCC 和 LLVM
VTV*	-fvtable-verify=std	GCC
CFI*	-fvisibility=default -fsantinize=cfi -flto -fuse-ld=gold	LLVM
全部防护	上述所有	
Asan*	-fsanitize=address –param=asan-stack=1	GCC 和 LLVM
无防护	-z execstack -fno-stack-protector (sysctl -w kernel_randomize_va_space=0)	GCC 和 LLVM

***：由于RISC-V架构的工具链原因，这些编译选项在RISC-V架构的编译器下并不被支持或是实现不完整**

测试结果总览



- GCC 10.3.0
- GLIBC 2.32
- LLVM 13

VTV/LLVM选项组

GCC VTV选项组

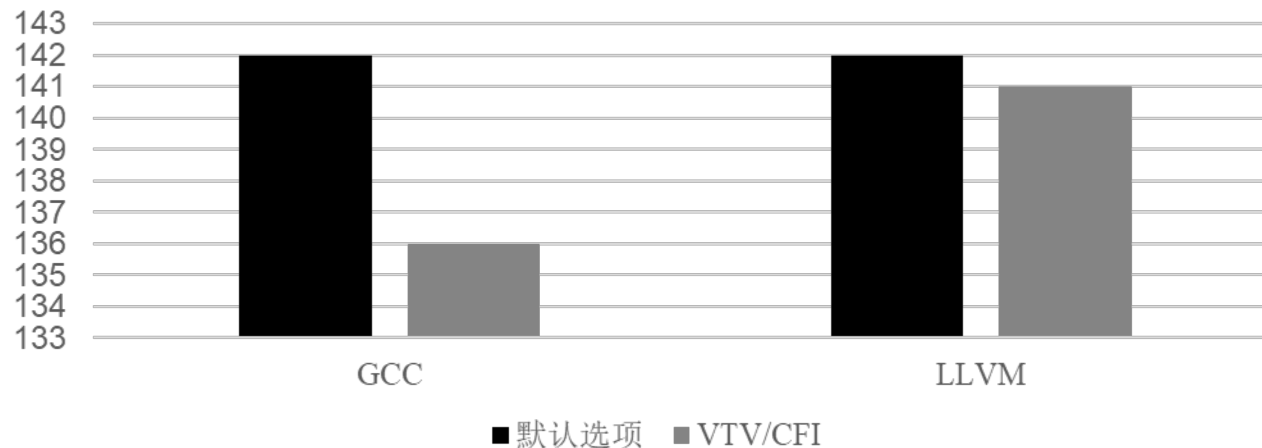
Intel Xeon 8280平台下:

- 6项伪造对象导向编程技术相关测例都执行失败
- 将虚函数表替换为子类、父类的行为没有被拦截

LLVM CFI选项组

Intel Xeon 8280平台下:

- 几乎没有提供任何安全增强
- 原因可能是库的动态链接方式



ASan选项组

Intel Xeon 8280平台下：

- GCC编译测试集**成功执行**的测例数为8
- LLVM编译测试集**成功执行**的测例数为21

测例	GCC	LLVM
函数体读取	成功执行	成功执行
全局偏移量表读取	成功执行	成功执行
全局偏移量表修改	执行失败 +	成功执行
栈上释放后使用	成功执行	执行失败 +
返回导向编程技术	执行失败	执行失败
伪造对象导向编程技术	执行失败 +	执行成功
跳转导向编程技术	执行失败 +	执行成功

结论2

- GCC编译器与LLVM编译器安全特性提供的内存安全检查大致相近
- 在使用动态链接类型定义时LLVM的CFI安全特性未能发挥有效作用，相比于GCC稍逊一筹.
- LLVM测试集中关于代码指针算数运算的测例没有通过编译，而GCC测试集中只是给出了警告，这也说明两款编译器对于内存安全问题防护具有不同的侧重点.
- 两款编译器提供的Address Sanitizer拦截了绝大多数的内存安全恶意行为，说明大多数的内存安全性质都依赖于内存的空间安全性.

后续工作

- 增加对其他主流ISA的支持
  - ARM AArch64
  - 龙芯/MIPS指令集架构的支持

- 测试集开源
  - https://github.com/comparch-security/cpu-sec-bench

致谢

- 感谢郭雄飞提供的HiFive Unleashed开发板.
- 感谢中国科学院软件所PLCT团队赠与的HiFive Unmatched开发板.

Q&A

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS