北大软件 | 打造富有生命力的软件
CREATE THE VITALITY OF SOFTWARE

北京大学 PEKING UNIVERSITY

基于静态分析的程序缺陷自动发现技术

马森

北京北大软件工程股份有限公司

2017年6月

内容提要

- 背景
- 静态分析技术
- 相关工具
- 实现技术及成果

背景及必要性

1978年，Hatford体育场倒塌

7000万美元

1985年，Therac-25放射治疗仪

三人死亡，三人重伤

1988年，蠕虫程序

蔓延互联网

1996年，Ariane火箭

损失5亿美元

2005年，EDS 儿童资助系统

损失约5亿多英镑

2006年，熊猫烧香

损失约1亿美元

软件灾难

OWASP历年安全问题Top10多数与**程序代码**相关

OWASP Top 10 (**2004**)
A1 – 未验证输入
A2 – 失效的访问控制
A3 – 失效的身份认证和会话管理
A4 – 跨站脚本 (XSS)
A5 – 缓冲区溢出
A6 – 注入漏洞
A7 – 不正确的错误处理
A8 – 不安全的加密存储
A9 – 拒绝服务
A10 – 不安全的配置管理

OWASP Top 10 (**2007**)
A1 – 跨站脚本 (XSS)
A2 – 注入漏洞
A3 – 恶意文件执行
A4 – 不安全的对象直接引用
A5 – 跨站请求伪造 (CSRF)
A6 – 信息泄露和不正确错误处理
A7 – 失效的身份认证和会话管理
A8 – 不安全的加密存储
A9 – 不安全的通信
A10 – 没有限制URL访问

OWASP Top 10 (**2010**)
A1 – 注入漏洞
A2 – 跨站脚本 (XSS)
A3 – 失效的身份认证和会话管理
A4 – 不安全的对象直接引用
A5 – 跨站请求伪造 (CSRF)
A6 – 安全配置错误
A7 – 不安全的加密存储
A8 – 没有限制URL访问
A9 – 传输层保护不足
A10 – 未验证的重定向和转发

OWASP Top 10 (**2013**)
A1 – 注入漏洞
A2 – 失效的身份认证和会话管理
A3 – 跨站脚本 (XSS)
A4 – 不安全的对象直接引用
A5 – 安全配置错误
A6 – 敏感信息泄露
A7 – 功能级访问控制缺失
A8 – 跨站请求伪造 (CSRF)
A9 – 使用含有已知漏洞的构件
A10 – 未验证的重定向和转发

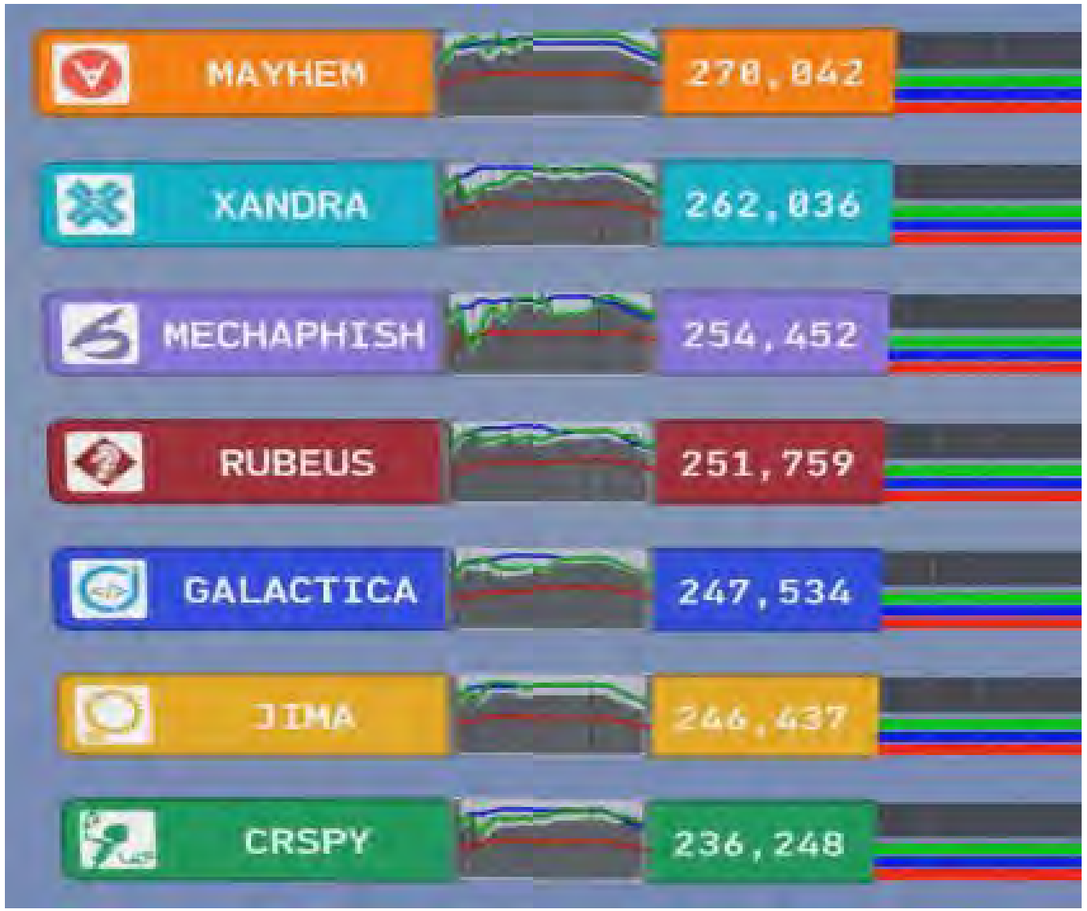
只有代码中的安全缺陷得以及时消除，最终形成的软件产品才能具备较高的安全性，有效降低整体系统安全风险

代码静态分析可以帮助用户从根源上减少30%-70%的软件安全问题

代码静态分析工具主要被国外产品所垄断，国内尚无成熟工具，存在安全隐患，无法做到自主可控

自动化漏洞发现比赛-CGC

CGC决赛
七支参赛队伍
每支队伍：
1280核
16TB内存
128TB存储空间



- 2016年8月美国DAPRA举办首届CGC挑战赛，目的是自动挖掘、利用及修复代码中的安全漏洞。
- 卡耐基梅隆大学Brumley团队的Mayhem夺得第一名
- MayHem在和人类的PK中处于劣势

在漏洞挖掘领域，机器远未像Alpha Go在智能搜索领域取得的成绩显著

自动化漏洞发现比赛-CGC

1st:
CMU
Mayhem

动态模块
污点分析
插桩代码
虚拟化执行

静态模块
符号执行
路径选择
利用生成
状态保存

2nd:
GrammaTech & Virginia Tech
Xandra

静态漏洞分析 ↔ 动态符号执行

静态漏洞分析 → 漏洞确认引擎

动态符号执行 → 漏洞确认引擎

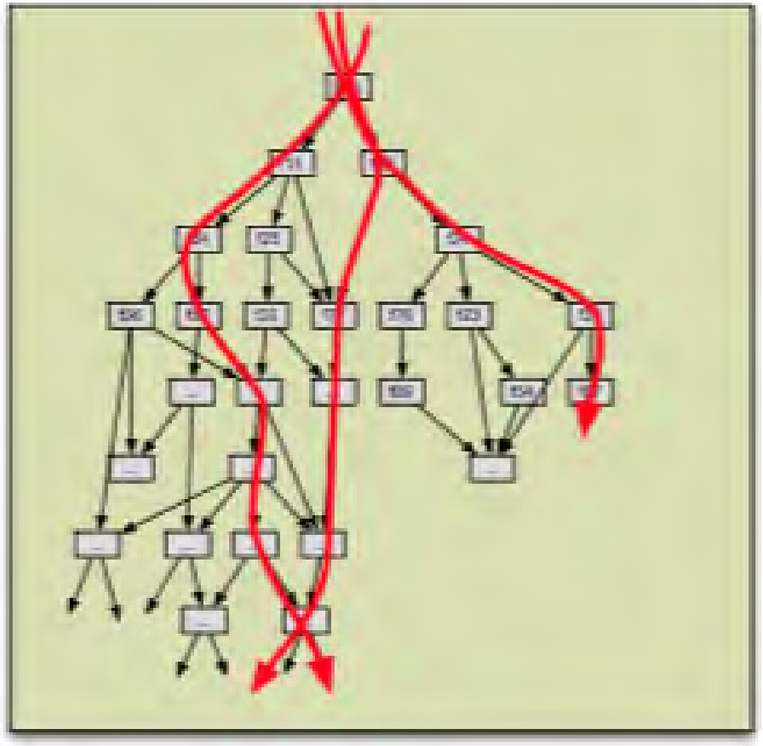
3rd:
UC-Santa Barbara
MechaPhish

模糊测试 → 动态分析

符号执行 ↔ 动态分析

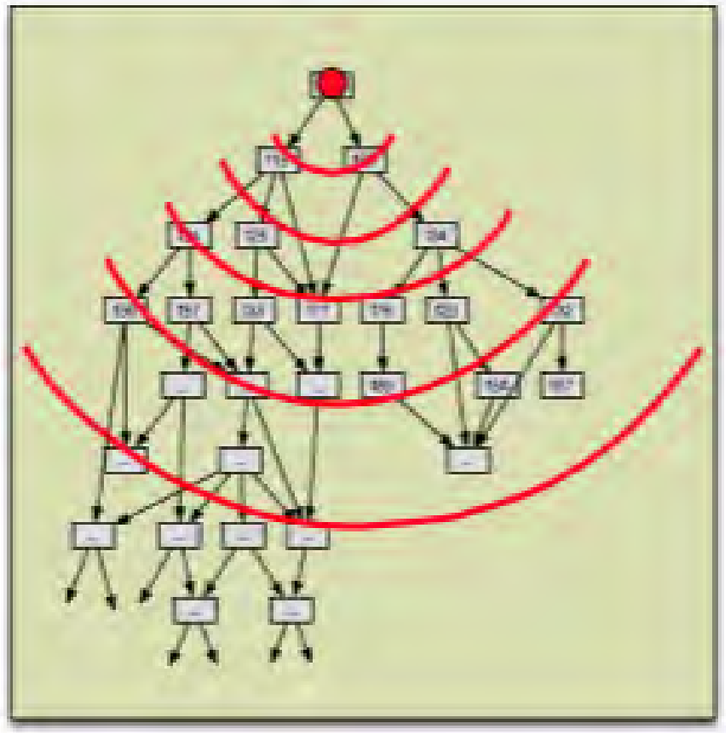
静态语义检测与动态测试的互补

动态测试



- 执行单一的路径
- 对关键的功能性测试很有效
- 对于Corner情况很难

静态语义检测

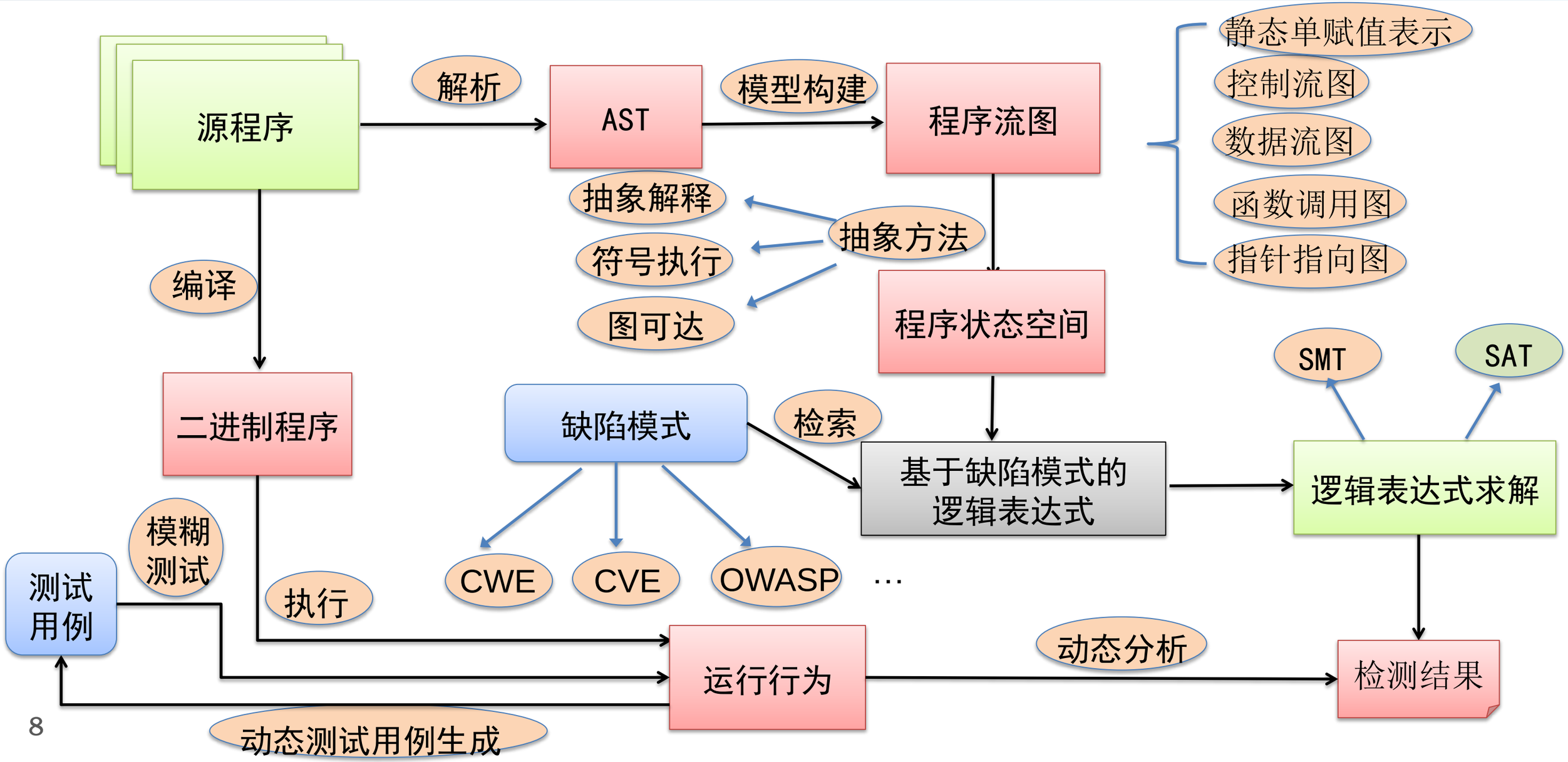


- 执行“符号”分析
- 对于具有缺陷模式的Corner情况非常有效
- 不能进行功能性测试

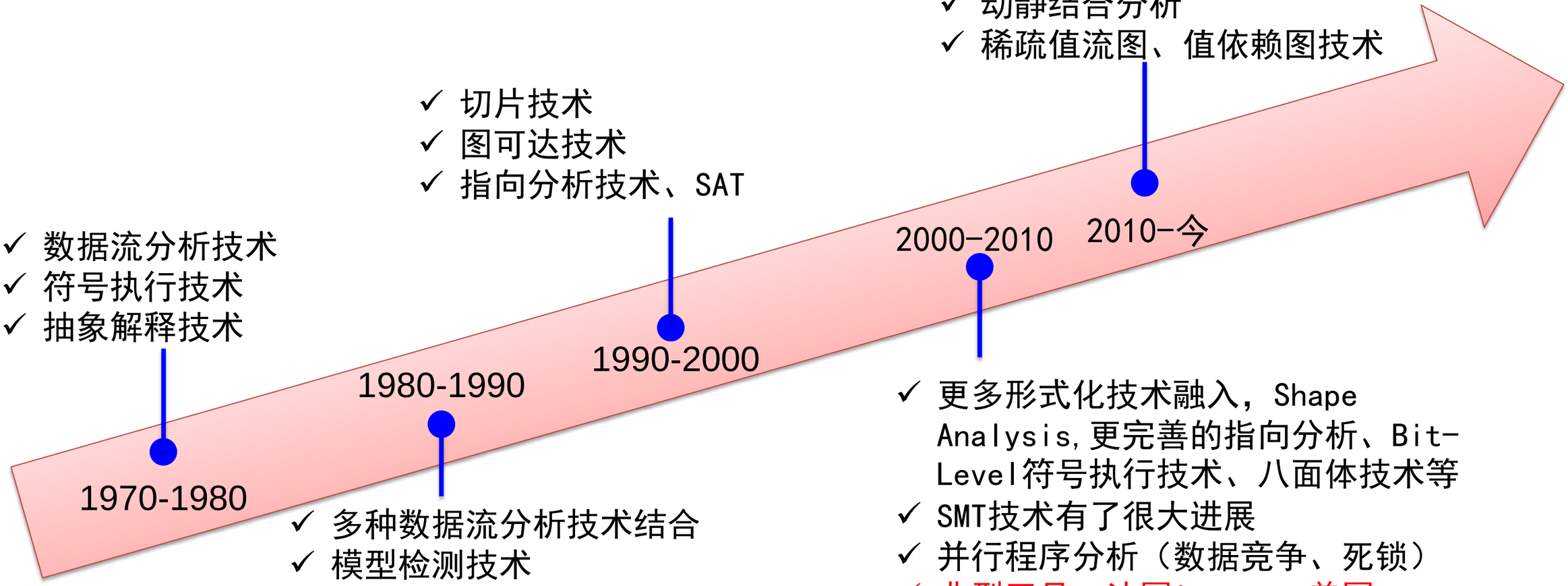
内容提要

- 背景
- 静态分析技术
- 相关工具
- 实现技术及成果

未知缺陷自动发现的基本流程



未知缺陷静态检测技术发展历程



未知缺陷自动发现的基本流程中相关概念

基本分析相关

Control Flow Graph（CFG）：控制流图

Call Graph（CFG）：函数调用图

SSA:静态单赋值

Super Graph：控制流图与调用图的

Mod-effect：修改影响分析，表达指针修改后对其他引用的影响

Dataflow Analysis：包括前后支配分析、到达定值分析、活跃变量分析等

继承关系分析：分析类之间的继承关系

Points-to：指向分析，表达程序中指针的指向关系

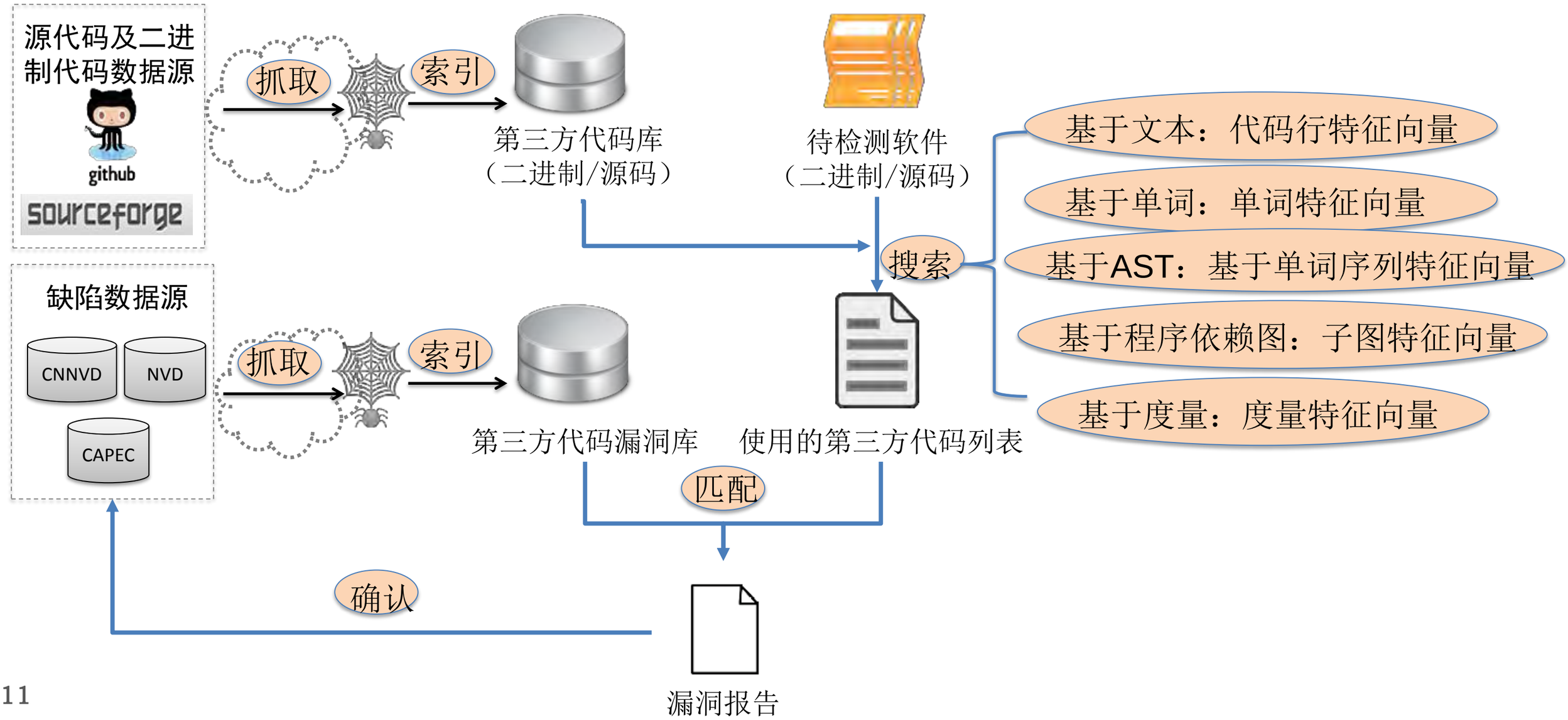
建立程序状态空间

- **抽象解释理论**是使用一个基于“抽象对象域”上的较低代价的计算过程来抽象逼近基于“受检程序指称对象域”上的计算过程，从而使得程序抽象执行的结果能够尽量反映出程序真实运行过程中的部分信息

符号执行是将程序源代码中变量的值采用抽象化符号的形式表示，模拟程序执行目的是分析程序中变量之间的约束关系，不需制定具体的输入数据，将变量作为代数中的抽象符号处理结合程序的约束条件进行推理，结果是一些描述变量间关系的表达式

图可达性分析根据分析问题的不同将程序应用一些函数分析技术或者算法转换为相应的图形，然后利用上下文无关语言的理论，计算点与点的可达性

基于同源分析的已知缺陷发现的基本流程



内容提要

- 背景
- 静态分析技术
- 相关工具
- 实现技术及成果

源代码漏洞自动发现工具

工具名称	研发机构	主要分析技术
Coverity Prevent	美国Synopsis公司	符号执行，模式匹配
Klocwork Insight	美国Rogue Wave Software公司	符号执行，区间分析
HP Fortify	美国HP公司	数据流分析，模式匹配
FindBugs	美国Maryland大学	模式匹配
Clang	美国UIUC	符号执行
cpplint	美国谷歌	模式匹配
Polyspace	美国MathWorks公司	抽象解释
Astree	法国巴黎LIENS实验室	抽象解释
Sparrow	韩国首尔国立大学	抽象解释

典型工具

国内工具：

- 北京大学 CoBOT
  - 基于值依赖图
  - 支持C/C++/Java语言
- 北京邮电大学 DTS
  - 基于符号执行的实现方式
  - 支持C/C++/Java语言

国外工具

- Synopsys Coverity
  - 基于符号执行的实现方式
  - 支持C/C++/Java/C#/Python语言

国内工具要么支持缺陷种类少，要么误漏报过高运行效率低

二进制漏洞自动发现工具

工具名称	研发机构	主要分析技术
Veracode	美国Veracode公司	禁运未知
Codesonar	美国GrammaTech公司	禁运未知
Vine	美国加州大学伯克利分校	数据流分析，符号执行
BAP	美国卡耐基梅隆大学	数据流分析，符号执行
Codesurfer/x86	美国威斯康辛大学	值集分析
McVeto	美国威斯康辛大学	模型检查
Bindead	德国慕尼黑理工大学	抽象解释
Angr	美国加州大学圣芭芭拉分校	符号执行
Jakstab	德国达姆施塔特理工大学	符号执行
Valgrind	开源社区（起源于英国剑桥）	代码插桩，污点分析
TEMU	美国加州大学伯克利分校	模拟执行，污点分析
Triton	法国Quarkslab	动态符号执行
Protecode	美国Synopsys公司	模式匹配

典型工具

- Bitblaze
  - 动静结合，基于Vine和TEMU
  - 效率较低，不支持某些指令集的解析
- Triton
  - 基于动态符号执行的测试用例生成
  - 难以应用于大型工程
- Protecode
  - 基于同源分析检测二进制代码漏洞
  - 不支持未知漏洞的检测

二进制漏洞检测工具在国内处于空白状态

静态测试工具应用：编码规则、缺陷与漏洞测试

国际标准

- **MISRA C/C++**
  - ✓ 由MISRA提出的C语言开发标准。目的在于提高嵌入式系统的安全性及可移植性
  - ✓ MISRA C 2004，分为21类，共计141项
  - ✓ ISO 17961 安全编码标准，45类
  - ✓ MISRA C++ 2008，分为21类，共计228项

国家标准

- **GJB 8114-2013**
  - ✓ 软件C/C++语言安全子集及编程规范
- **GJB 5369-2005**
  - ✓ 由航天科工集团公司提出，并结合航天型号软件特点，经裁减补充而成。分为15类，共计138项

行业标准

- **921 C-2007**
  - ✓ 依据GJB 5369，制定了航天“921”工程C语言软件编程准则和要求。分为15类，共计163项
- **BACC C-2008**
  - ✓ 航天某单位C语言编程规范，分为17类，共计134项
- **CRSC C-2014**
  - ✓ 高铁领域C语言编程规范，分为18类，共计218项

指针缺陷类型\安全等级	Critical (1级)	Severe (2级)	Error (3级)
内存泄漏		4	
资源泄漏		1	
空指针解引用	16		
释放非堆内存	1	1	
释放未分配的内存	2		
使用已释放的内存	4	4	
函数返回局部变量的地址	3		
未初始化使用	4	1	
释放不成功			1
合计	30	11	1

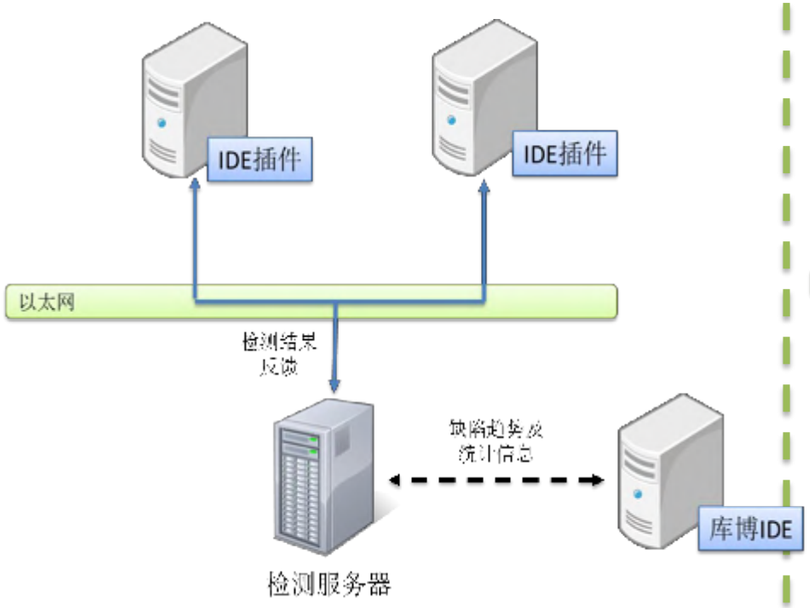
其他缺陷类型\安全等级	Critical (1级)	Severe (2级)	Error (3级)
不可达路径	2	1	
非法计算			4
库函数(打印)			5
库函数(网络访问)		4	20
库函数(输入输出)		10	22
库函数(本地访问)		1	6
库函数(宏)			4
合计	2	16	61

CWE缺陷

- OWASP中活跃且严重性等级较高的漏洞：
  - ✓ 缓冲区溢出（27种）
  - ✓ 数组越界 （24种）
  - ✓ 整数溢出（15种）
  - ✓ 浮点数溢出（15种）
  - ✓ SQL注入（10种）

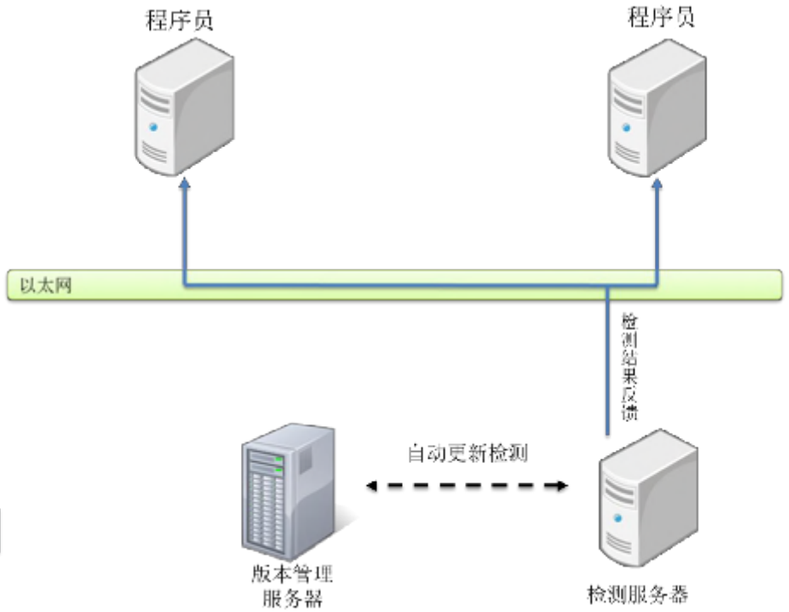
静态测试工具应用场景（以库博为例）

开发人员应用



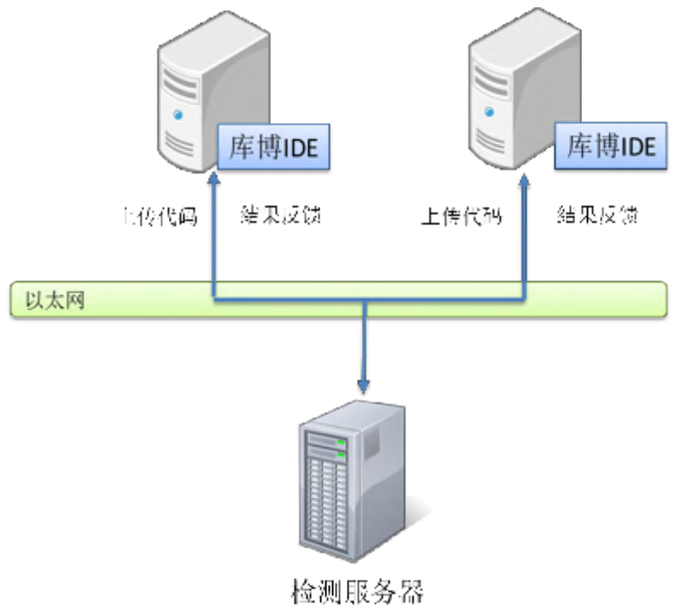
采用IDE插件的方式，与 VC 、 Tornado 、KMC等开发工具集成

持续集成应用



每天定时从版本服务器中更新、检测，第二天将结果回送

测试人员应用



利用管理界面上传不同工程的代码、随时进行分析检测

测试结果

- 为了验证编码规则和程序缺陷检测方面的效果，需要进行三种类型的测试

中等规模开源项目，测试误报率和漏报率

选择SPEC 2000（由标准性能评价组织SPEC开发的用于评测通用型CPU性能的基准程序测试组）中十个开源项目进行验证分析，项目规模在万行至几十万行之间，主要目标是测试COBOT的误报率和漏报率

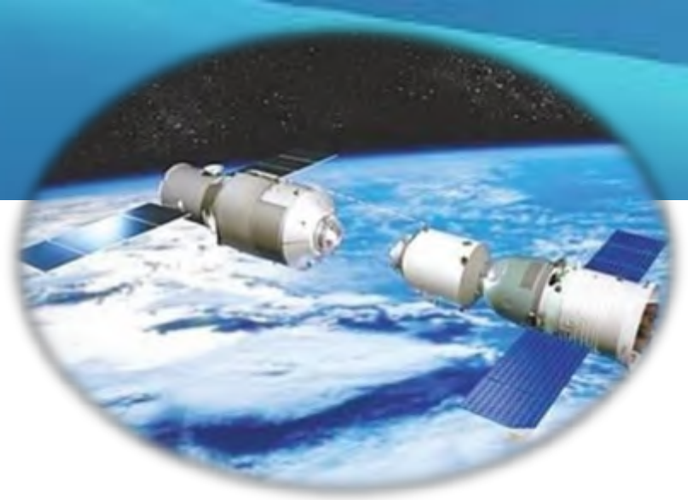
较大规模开源项目，测试运行效率及误报率

选择规模在三个百万行以上的大型开源项目，主要目标是测试工具的运行效率及误报率

实际应用项目，综合评估实际运行效果

将工具部署到实际运行环境中，对约数十万行代码进行检测，综合评估实际运行效果。

航天某单位应用比较：程序缺陷分析



项目		CoBOT（库博）		Klocwork		相同缺陷			
名称	代码行	检测时间	误报数/总数	检测时间	误报数/总数	合计	资源泄漏	空指针解引用	使用释放内存
1	4108	20.2	0/4	48.1	0/2	2	2	0	0
2	3715	21.1	2/5	44.4	0/3	3	3	0	0
3	1287	10.5	0/3	38.2	0/3	3	0	3	0
4	331	10.8	0/0	28.1	0/0	0	0	0	0
5	3641	13.5	0/0	63.2	0/0	0	0	0	0
6	2744	14.8	0/0	61.3	0/0	0	0	0	0
7	785	10.8	0/0	31.3	0/0	0	0	0	0
8	3304	21.5	0/1	51.4	0/0	0	0	0	0
9	7339	16.1	0/0	21.2	0/0	0	0	0	0
10	14480	20.7	0/0	31.6	0/0	0	0	0	0
11	8515	17.2	0/2	42.6	0/0	0	0	0	0
12	2417	20.7	0/6	49.2	0/6	6	6	0	0
13	2669	21.9	0/8	45.3	0/8	8	8	0	0
14	3457	32.2	1/9	56.4	0/8	8	8	0	0
15	2173	22.6	0/6	50.3	0/6	6	6	0	0
16	4158	20.5	0/0	45.2	0/0	0	0	0	0
17	3251	23.1	0/0	43.5	0/0	0	0	0	0
18	1650	13.3	0/2	65.3	0/2	2	2	0	0
19	1912	11.1	0/3	43.2	0/3	3	3	0	0
20	1235	13.2	0/0	35.5	0/0	0	0	0	0
21	1621	10.2	1/1	65.5	0/0	0	0	0	0
22	537	10.8	0/0	35.4	0/0	0	0	0	0
23	14254	20.3	0/1	42.8	0/0	0	0	0	0
总计	89583	397.1	4/51	1039	0/41	41	38	3	0

CoBOT（库博）发现了6个Klocwork未发现的缺陷：

- 项目23：空指针解引用 1个；
- 项目11：资源泄漏 2个；
- 项目8：资源泄漏 1个；
- 项目1：资源泄漏 2个

漏报率：CoBOT在10%左右，Klocwork在20%以上
误报率：CoBOT在10%-20%之间，Klocwork低于10%

轨道某单位应用比较：编码规则检测

CoBOT		Testbed	
漏报率	误报率	漏报率	误报率
1.32%	0.06%	2.60%	0.32%

误报率：相比于CoBOT，Testbed误报率高出0.26%，漏报率高出1.28%。可见，误漏报率均高于CoBOT

ID	描述	Testbed漏报	Testbed误报	CoBOT漏报	CoBOT误报
37S	过程参数只有类型没有标识符	2	0	0	0
110S	使用单行注释//	34	0	0	0
130S	使用了不允许使用的 Included 文件	7	0	0	0
331S	字面值需要一个 u 后缀	14	0	67	0
58S	存在空语句	18	0	0	4
59S	if 语句中没有 else 分支	100	0	0	0
62D	指针参数应被定义为常量	74	1	6	0
67S	在模块中使用#define	38	0	0	0
68S	使用#undef	2	0	0	0
74S	避免使用联合	2	0	0	0
87S	用指针进行代数运算	16	0	0	0
119S	使用嵌套注释	0	2	0	0
21S	……	……	……	……	……
总计		323	40	164	7

航空某单位应用比较：编码规则检测

- 采用GJB 5369编码规则，对襟缝翼系统（代码行数为22051行）进行了比对测试
- 比对对象是Testbed Ver7.8.3

Testbed测试较差的条目：

编号	描述	Testbed 误报率	Testbed 漏报率	COBOT 误报率	COBOT 漏报率
1.2.4	在结构体定义中使用位域	100.00%	100.00%	0.00%	0.00%
14.1.1	禁止对浮点类型的变量做相等比较操作	100.00%	100.00%	0.00%	0.00%
8.1.3	用八进制数	100.00%	100.00%	0.00%	0.00%
1.1.16	对变量重命名	100.00%	100.00%	0.00%	0.00%
2.1.4	逻辑上关联的表达式需要括号	100.00%	100.00%	0.00%	0.00%
10.2.1	注释中可能包含代码	35.56%	57.35%	1.45%	0.00%
6.1.12	对有符号类型使用位操作	0.00%	98.33%	53.13%	1.64%
6.1.9	符号型/无符号型之间没有使用强制类型转换	0.00%	51.60%	0.00%	0.00%

CoBOT		Testbed	
漏报率	误报率	漏报率	误报率
1.93%	6.06%	18.91%	25.01%

误报率：相比于CoBOT，Testbed误报率高出18.95%，漏报率高出16.98%。可见，误漏报率均远高于CoBOT

内容提要

- 背景
- 静态分析技术
- 相关工具
- 实现技术及成果

值依赖分析技术

值依赖分析技术是一种符号计算技术，通过值依赖分析构建的值依赖模型：

- 表达了程序中全部元素，并建立了的6类值依赖关系及11种值依赖关系

Dependency	Node	Added Edges
Flow	n: $y = x$ $n_w(x)$: each node that writes x	$n_w(x) \rightarrow_x n$
Store	n: $*x = y$ $n_u(x)$: each node that uses x directly, such as y=x or y=f(x)	$n \mapsto_x n_u(x)$
	n: each node that contains &x $n_w(x)$: each node that writes x	$n_w(x) \mapsto_{\&x} n$
Load	n: each node that contains $*x$ $n_w(x)$: each node that writes x	$n_w(x) \dashrightarrow_{*x} n$
	n: each node that contains &x $n_u(x)$: each node that uses x directly, such as y=x or y=f(x)	$n \dashrightarrow_x n_u(x)$
Call	n: $r = f(\dots a_k \dots)$ $n_f(a_k)$: formal parameter n_{ap_k}: actual parameter	$n_f(a_k) \hookrightarrow_f n_{ap_k}$
Return	$r = f(\dots a_k \dots)$ n_{ret}: return node n_r: the node that writes r	$n_{ret} \hookrightarrow_f n_r$
Entry	If a guard exists from entry to definition nodes n_d	$Entry \mapsto_x n_d$

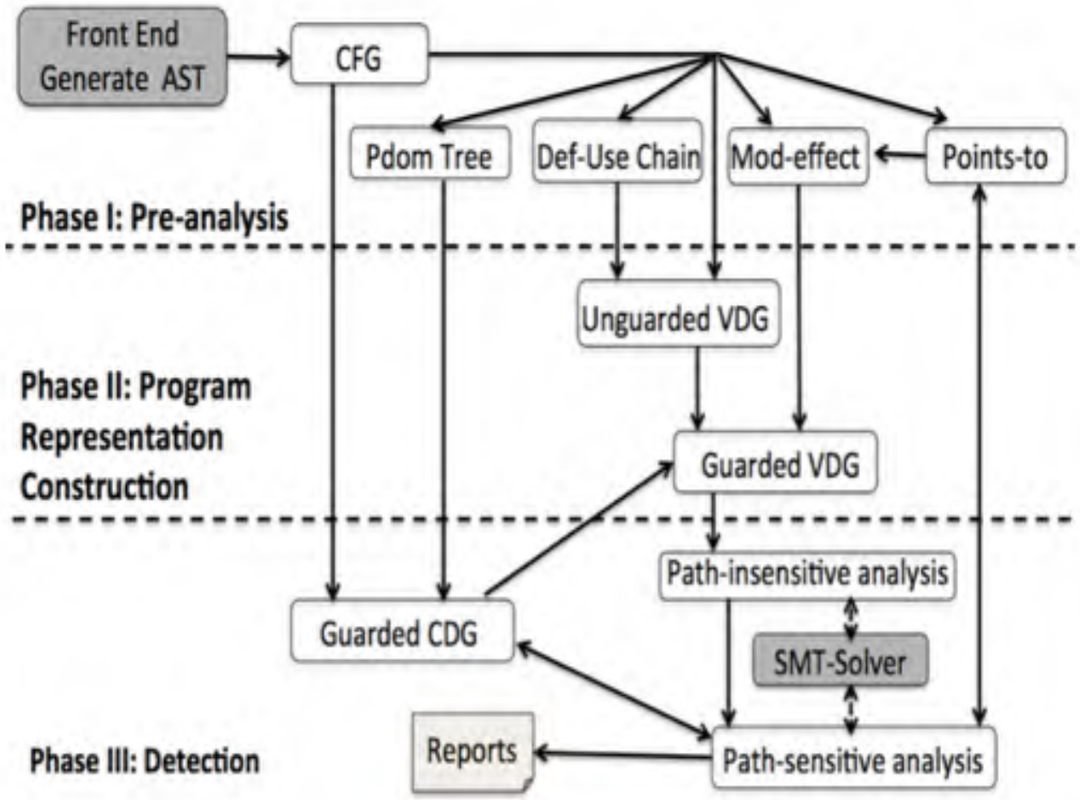
- 综合利用内存的指向信息、变量的可能取值范围信息、增强了模型对程序值依赖描述的准确性
- 通过遍历CFG并利用SSA信息，计算了变量值之间的约束关系，作为值依赖的守卫表达式

值依赖图是一个有向图，由以下四元组组成：

$$\langle N_d^x, \partial, \varphi, N_s \rangle$$

- 其中 N_d^x 表达了X的定义点集合
- ∂ 表达了定义点X到其依赖点的映射及值依赖关系见左图，即 $N_d^x \Rightarrow N_u^x$.
- φ 表达了依赖关系所对应的守卫表达式即 $N_d^x \overset{\varphi}{\Rightarrow} N_u^x$,
- N_s 表达了 φ 所对应的选择语句集合

值依赖分析技术



相对以符号执行为主的解决方案：

1、检测模型更加简化，以变量值的角度构建模型而不是以控制流的方式，这样可以省去与值变化无关的点，使分析效率更高

2、对于精度相对于符号执行，其只能按照符号执行的逻辑不断计算有可能发生状态爆炸，而值依赖计算可以根据程序的复杂度设定迭代次数，是一种主动的折衷方案

3 值依赖模型在由于是跨函数的没有摘要的步骤，所以对于跨函数的缺陷模式精度更高，函数内精度相仿

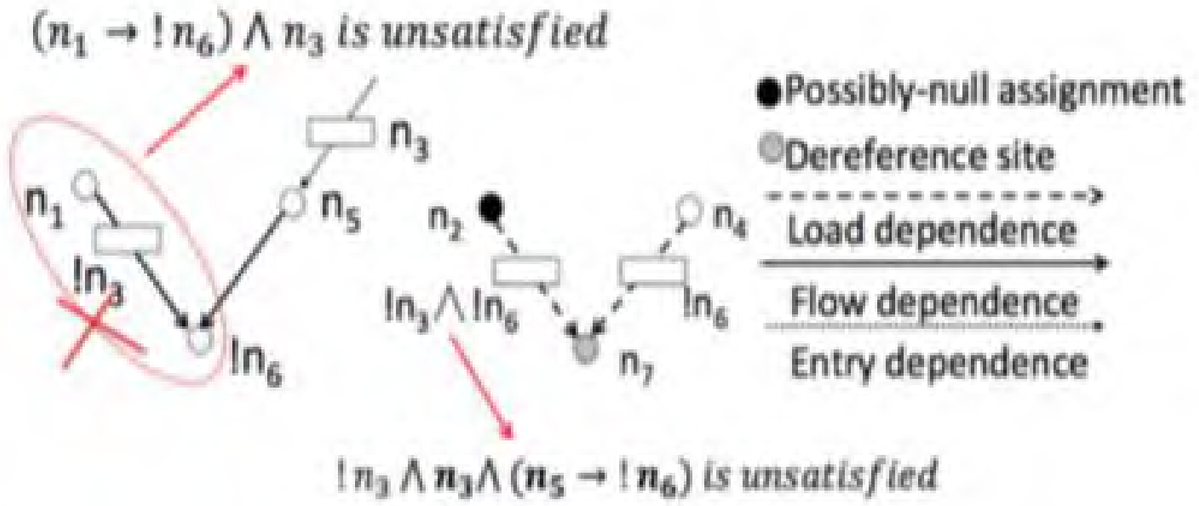
值依赖模型与约束求解：空指针解引用

➢空指针解引用的实例

典型的Infeasible Path程序

```
1   int lastAtomIsMoov = 0;                    //n1
2   MP4Atom* pLastAtom = NULL;                 //n2
3   ...
4   if(*pFirstAtom > 0){                       //n3
5     pLastAtom = &pAtom;                      //n4
6     lastAtomIsMoov = 2;                      //n5
7   }
8   ...
9   if(lastAtomIsMoov < 2){                    //n6
10  ...
11  }
12  else{
13    SetPosition(pLastAtom->getEnd());        //n7
14  }
```

值依赖模型及一次迭代求解过程

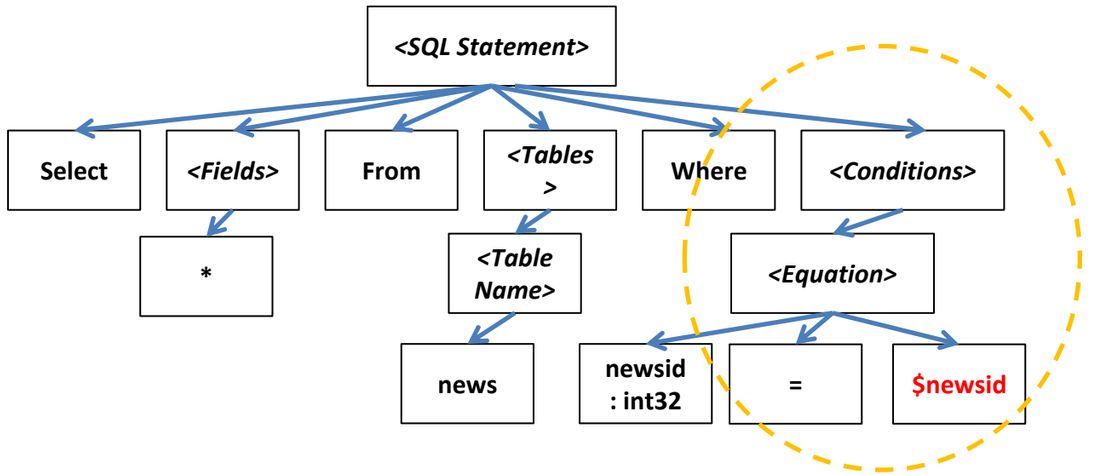


经过一次迭代，约束表达式由原来的$!n_3 \wedge !n_6$转换成了$!n_3 \wedge n_3 \wedge (n_5 \rightarrow n_6)$，得到了Unsatisfied结果，从而消除了误报

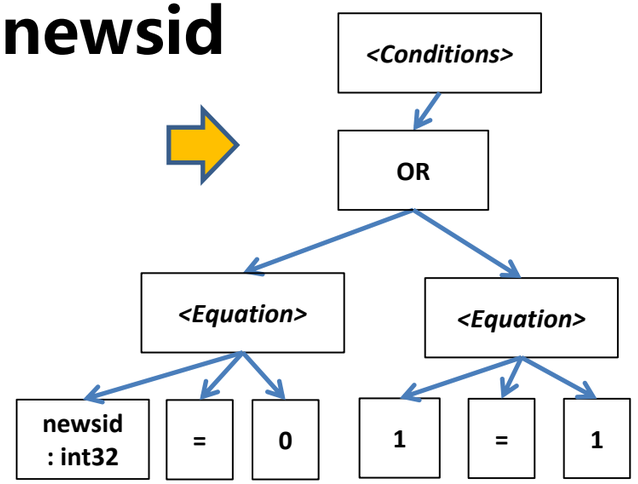
字符串约束求解：SQL注入类漏洞

➢字符串约束求解实例

```
1    $username = ‘posted_user’;
2    $newsid = $_POST[‘posted_newsid’];
     ……
6    if (!preg_match(‘/[\d]+$/’, $newsid )) {
7          Handle exception…
8          exit;
9      }
     ……
16   $newsid = $DB->query("SELECT * FROM ‘news’
                 WHERE newsid=".$newsid);
```

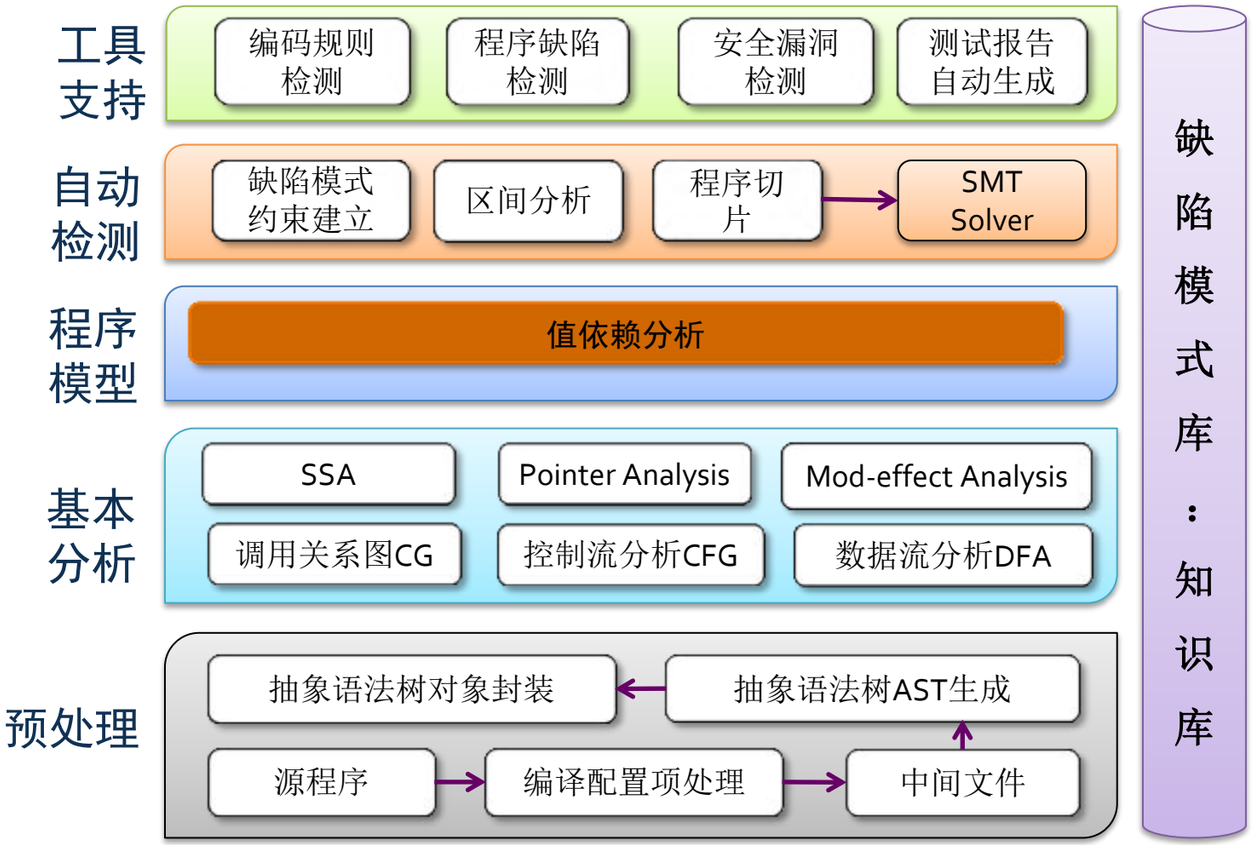


自动生成符号变量$newsid
SQL注入攻击向量
0 or 1=1;#0



分析Web应用源代码漏洞，自动生成攻击向量

基于值依赖分析技术的静态分析工具：库博(COBOT)



- COBOT是一款代码静态分析工具
  - ✓ 采用基于专利技术分析引擎开发的、具有自主知识产权程序静态分析框架，综合运用了多种先进的静态分析技术
  - ✓ 及时发现代码问题，自动识别程序缺陷、安全漏洞和编码风格问题
  - ✓ 并可以进行软件度量及规则的定制化分析、质量报表的模板定制

编码规则检测（10类标准，1000+条规则）

程序缺陷分析（CWE，14类110+种）

安全漏洞扫描（OWASP，8类90+种）

相关成果

- 相关技术成果发表在国际顶级会议和国内刊物上
  - ✓Safe Memory-Leak Fixing for C Programs, ICSE’15
  - ✓Fixing Recurring Crash Bugs via Analyzing Q&A Sites, ASE’15
  - ✓Practical Null Pointer Dereference detection via graph reachability, ISSRE’15
  - ✓基于值依赖分析的空指针解引用，电子学报
- 获得软件著作权8项，申请专利2项
- 2015年，北京市/国家科技创新优秀成果奖
- 中国唯一一个获得CWE Compatible的安全产品

静态测试技术的未来前景

- 每小时千万行以上的程序静态缺陷扫描技术
  - 程序规模不短增大，持续集成要求分析效率与编译速度同一量级，
- 缺陷模式的全自动总结及挖掘技术
  - 如何利用机器学习技术自动挖掘已知的漏洞模式
- 跨语言分析技术，尤其是对于复杂MVC配置的大型工程
  - 大型系统采用MVC框架，通过配置文件确定数据传输处理的程序
- 动静结合的漏洞自动分析方法的不断完善
  - 采用人工智能技术克服静态分析效率问题以产生更多有效的测试用例


汇报完毕，谢谢！