PatchScope: Memory Object Centric Patch Diffing(CCS'20)

Lei Zhao[1], Yuncong Zhu[1], Jiang Ming[2], Yichen Zhang[1], Haotian Zhang[2], Heng Yin[3]
[1]Wuhan University, China
[2] University of Texas at Arlington, USA
[3] University of California, Riverside, USA

报告人：王笑克
武汉大学国家网络安全学院
2021年6月6日

背景

- 软件补丁是漏洞修复的重要机制

- 补丁携带了安全相关的信息
  - 自身带来的安全影响
  - 对应漏洞的细节

背景

- 补丁携带了安全相关的信息
  - 自身的安全影响
  - 对应漏洞的细节

- 利用补丁携带的安全信息
  - 更快了解对应程序的脆弱点
  - 评估是否是 buggy patch
  - 挖掘 “n-day” 漏洞 & 构造类似安全补丁

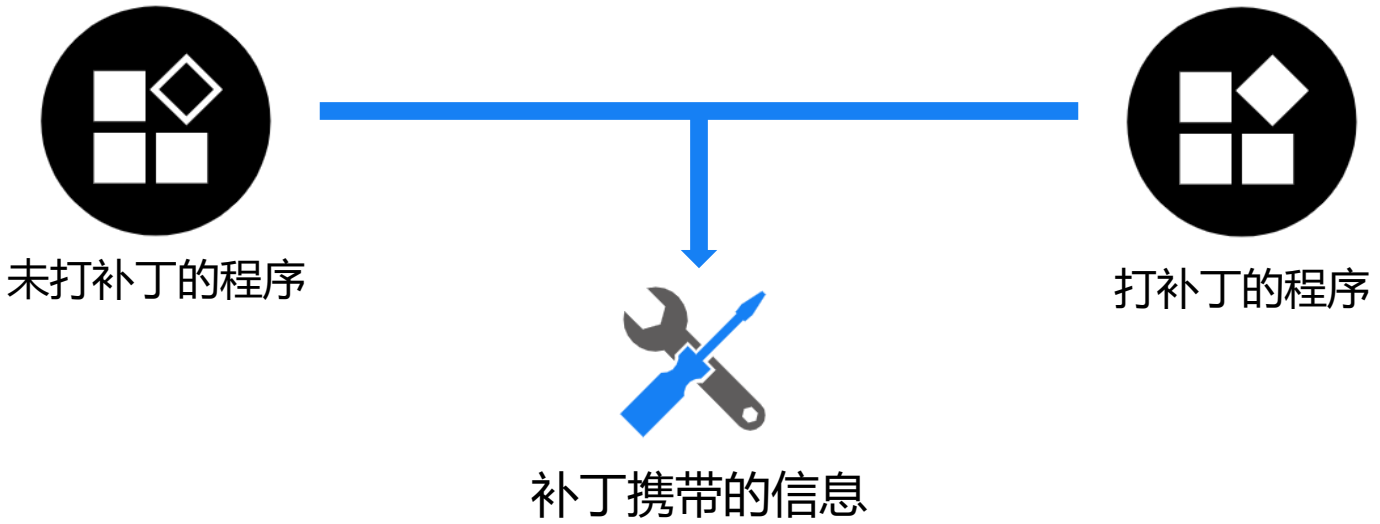
背景

- 补丁携带的安全信息往往难以轻易获取
- 软件供应商
  - 在更新公告中掩盖补丁细节
  - 闭源软件直接以二进制形式发布

- 漏洞数据库
  - 漏洞公告只提供漏洞基本信息（如漏洞类型和安全影响）
  - 漏洞的 root cause 与其对应补丁获取不全

研究问题

- 二进制补丁差异性比对
  - 在二进制层面精确识别补丁带来的差异
  - 为补丁细节和修复的漏洞提供丰富信息



现有工作

基于语法的二进制比对技术	基于符号执行的二进制比对技术	基于语义感知的二进制比对技术	基于人工智能的二进制比对技术
• BinDiff (2011) • DarunGrim (2011) • Diaphora (2015)	• BinHunt (ICICS'08) • DiSE (PLDI'11) • CoP (ASE'14) • Esh (PLDI'16)	• BLEX (USENIX'14) • IMF-SIM (ASE'17) • BinSim (USENIX'17)	• Genius (CCS'16) • InnerEye (NDSS'19) • Asm2vec (S&P'19) • DeepBinDiff (NDSS'20)

现有工作

基于语法的二进制比对技术
• BinDiff (2011) • DarunGrim (2011) • Diaphora (2015)

- 基于调用图/控制流图进行相似性比较
- 使用启发法匹配函数和基本块
- 结构、指令敏感

- 用于补丁比对时的缺点：
  - 当出现新的函数节点时或指令对齐有误时会导致差异指令数量剧增

现有工作

基于符号执行的二进制比对技术

- BinHunt (ICICS'08)
- DiSE (PLDI'11)
- CoP (ASE'14)
- Esh (PLDI'16)

- 对二进制代码进行符号执行
- 使用定理证明器比较相似性

- 用于补丁比对时的缺点：
  - 输出的符号公式无助于理解补丁信息
  - 难以应对库函数、系统调用等涉及运行环境交互的代码

现有工作

基于语义感知的二进制比对技术

- BLEX (USENIX'14)
- IMF-SIM (ASE'17)
- BinSim (USENIX'17)

- 使用系统调用、库函数表示语义信息
- 缺点：粒度太粗，对微小的差异不敏感

- 将二进制代码视为黑盒，使用动态测试来对行为进行相似性度量
- 缺点：粒度太细，会识别出较多差异

现有工作

基于人工智能的二进制比对技术
• Genius (CCS'16) • InnerEye (NDSS'19) • Asm2vec (S&P'19) • DeepBinDiff(NDSS'20)

- 将二进制代码转换成嵌入向量
- 使用 NLP 技术提取语义信息
- 使用机器学习算法进行相似度计算

- 用于补丁比对时的缺点：
  - 对微小的差异不敏感
  - 不足以表述补丁差异

当现有方法用于补丁差异比对

- 现有的二进制比对方法
  - 大多基于控制流结构与指令层面进行分析
  - 往往倾向于容忍一定程度的代码变动
    - 编译器版本、编译优化选项、指令集……
    - 好处：能够更好地适应一些二进制代码的变种
- 无法有效识别不影响控制流结构/指令的变化
- 难以准确捕获 patch 带来的微小改变

评价是否相似 ≠ 找差异

```
if (argv[i][0] == '-') {
    if (!strncmp(argv[i], "-prestring", 10)) {
        nflags++;
-       ret = sscanf(argv[i] + 1, "prestring=%s", buf);
+       ret = sscanf(argv[i] + 1, "prestring=%490s", buf);
        if (ret != 1) {
            fprintf(stderr, "parse failure for prestring\n");
            return 1;
        }
```

CVE-2018-7186 补丁源代码

08048E3C	mov	eax, ss:[ebp+command]	08048E3C	mov	eax, ss:[ebp+command]
08048E3F	mov	ss:[esp+4], 0x80493A6	08048E3F	mov	ss:[esp+4], 0x80493A6
08048E47	mov	ss:[esp+8], eax	08048E47	mov	ss:[esp+8], eax
08048E4B	mov	eax, ss:[ebp+str]	08048E4B	mov	eax, ss:[ebp+str]
08048E4E	mov	eax, ds:[eax+ebx*4]	08048E4E	mov	eax, ds:[eax+ebx*4]
08048E51	add	eax, bl 1	08048E51	add	eax, bl 1
08048E54	mov	ss:[esp], eax	08048E54	mov	ss:[esp], eax
08048E57	call	. __isoc99_sscanf	08048E57	call	. __isoc99_sscanf

BinDiff 二进制对比结果

补丁代码模式

- 对安全补丁的代码模式进行了大规模调研
- 总结了 2205 个内存破坏类漏洞补丁的模式

补丁代码模式及类型非常复杂

No.	Category	Percentage
1	add input sanitization checks	43.5%
2	change input sanitization checks	25.1%
3	add data structures	6.1%
4	change data structure definitions	6.5%
5	change data structure references	22.3%
6	change function parameters	10.9%
7	add or change function calls	15.3%
8	add functions	4.7%
9	change functions	7.6%

截断不安全的输入

修改数据结构

修复漏洞所在函数

补丁代码模式所带来的代码改动

- 对安全补丁的代码模式进行了大规模调研
- 总结了 2205 个内存破坏类漏洞补丁的模式

No.	Category	Percentage
1	add input sanitization checks	43.5%
2	change input sanitization checks	25.1%
3	add data structures	6.1%
4	change data structure definitions	6.5%
5	change data structure references	22.3%
6	change function parameters	10.9%
7	add or change function calls	15.3%
8	add functions	4.7%
9	change functions	7.6%

控制流产生变化

补丁代码模式所带来的代码改动

- 对安全补丁的代码模式进行了大规模调研
- 总结了 2205 个内存破坏类漏洞补丁的模式

No.	Category	Percentage
1	add input sanitization checks	43.5%
2	change input sanitization checks	25.1%
3	add data structures	6.1%
4	change data structure definitions	6.5%
5	change data structure references	22.3%
6	change function parameters	10.9%
7	add or change function calls	15.3%
8	add functions	4.7%
9	change functions	7.6%

控制流未产生变化

补丁代码模式所带来的代码改动

- 对安全补丁的代码模式进行了大规模调研
- 总结了 2205 个内存破坏类漏洞补丁的模式

No.	Category	Percentage
1	add input sanitization checks	43.5%
2	change input sanitization checks	2.7%
3	add data structures	6.1%
4	change data structure definitions	6.5%
5	change data structure references	22.3%
6	change function parameters	10.9%
7	add or change function calls	15.3%
8	add functions	4.7%
9	change functions	7.6%

如何在适应不同补丁模式的同时提供丰富的语义信息?

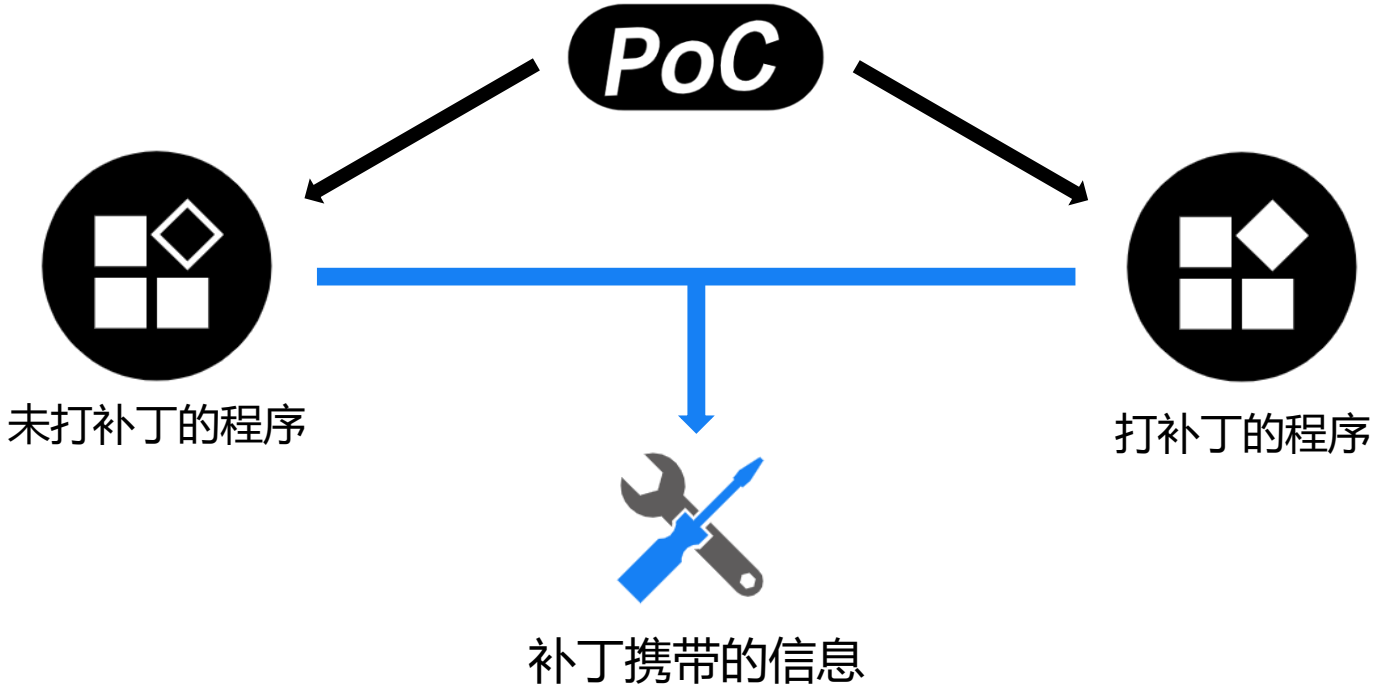
不会引起指令的变化

仅关注控制流及指令层面难以充分描述补丁带来的差异

观察与动机

- 程序对输入的操作能够揭示丰富的语义
- 安全补丁通过修改对数据结构的操作来规范输入的传播

=>怎样揭示运行时数据结构访问与输入字段之间的关系



内存对象访问序列

- 定义1：内存对象
- mobj = (alloc, size, type)
  - alloc：内存对象分配时的上下文
  - size：内存对象的大小
  - type：内存对象的类型
    - 静态变量
    - 栈中的局部变量
    - 堆中的动态变量

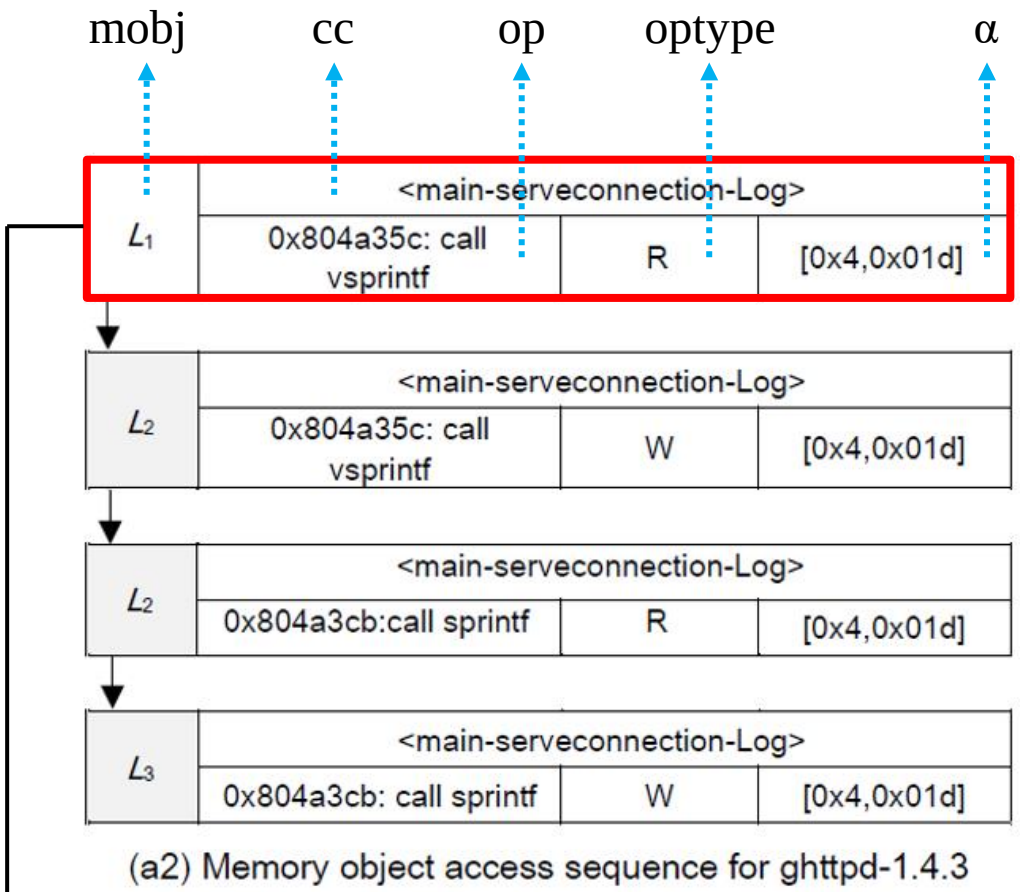
mobj	*alloc*	*size*	*type*
L_1	<main-serveconnection>:ebp-0x4151	0x2000	stack
L_2	<main…Log>: ebp-0x172	0xc8	stack
L_3	<main…Log>: ebp-0x23a	0xc8	stack

(a1) Memory object representation for ghttpd-1.4.3

程序在 **main-serverconnection** 函数栈的 **ebp-0x4151** 处分配了大小为 **0x2000** 的内存对象 **L1**

内存对象访问序列

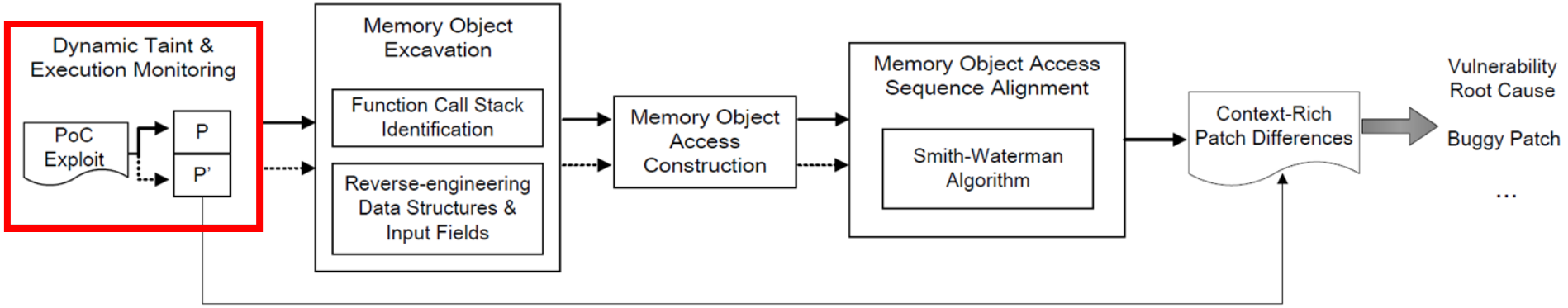
- 定义2：内存对象访问
- A(mobj) = (mobj, cc, op, optype, α)
  - mobj：内存对象
  - cc：内存对象访问的函数上下文
  - op：访问内存对象时正在执行的操作
    - 数据移动指令
    - 算术指令
    - 库/系统调用的调用指令
  - optype：访问的类型（read/write）
  - α：内存对象相关的输入字段



(a2) Memory object access sequence for ghttpd-1.4.3

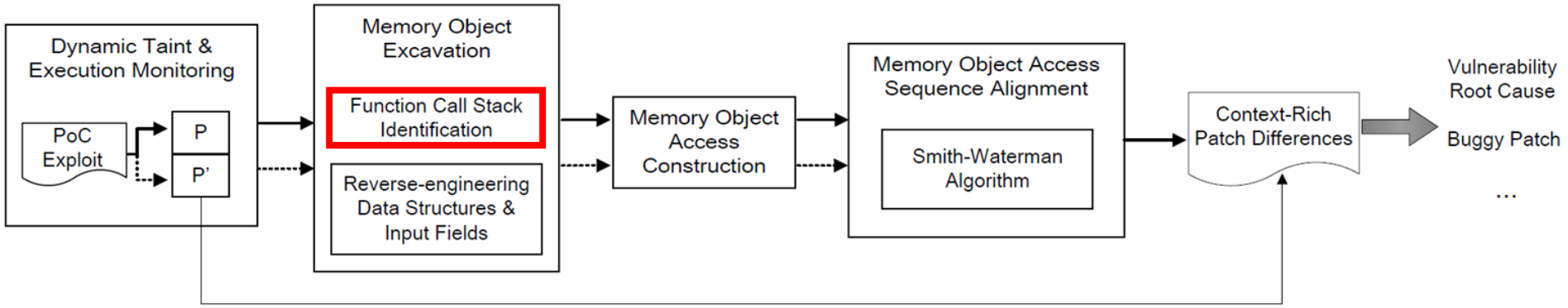
程序在 **main-serverconnection-Log** 函数中执行 **call vsprintf** 时**读取**了内存对象 **L1**
其中，L1 受输入字段 **[0x4,0x1d]** 字节影响

工作流程



- 第一步：动态污点分析和执行监视
  - 细粒度运行时信息
  - 多源污点分析
- => 得到执行序列 + 输入字节的污点标签传播情况

工作流程

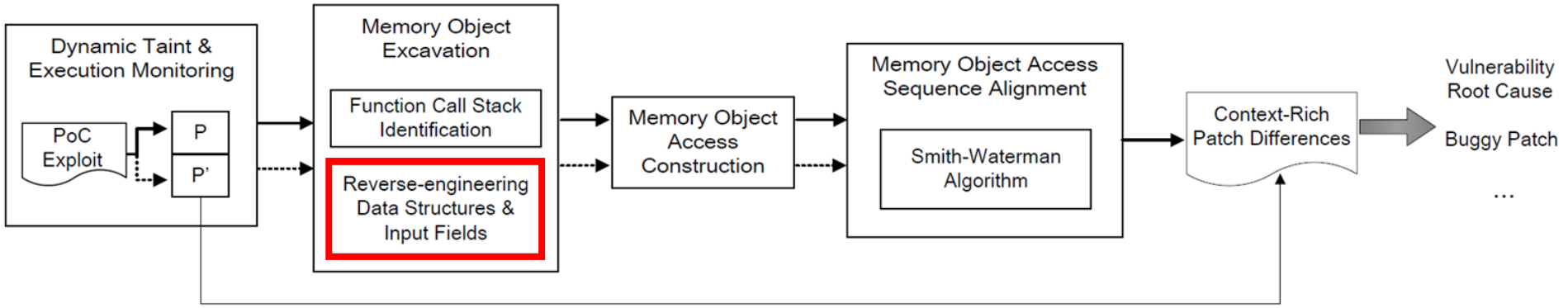


- 第二步：内存对象提取
  - 函数调用栈识别
    - 识别成对的 call/ret 指令
    - 尾调用
    - 函数内联

mobj = (alloc, size, type)

A(mobj) = (mobj, cc, op, optype, α)

工作流程

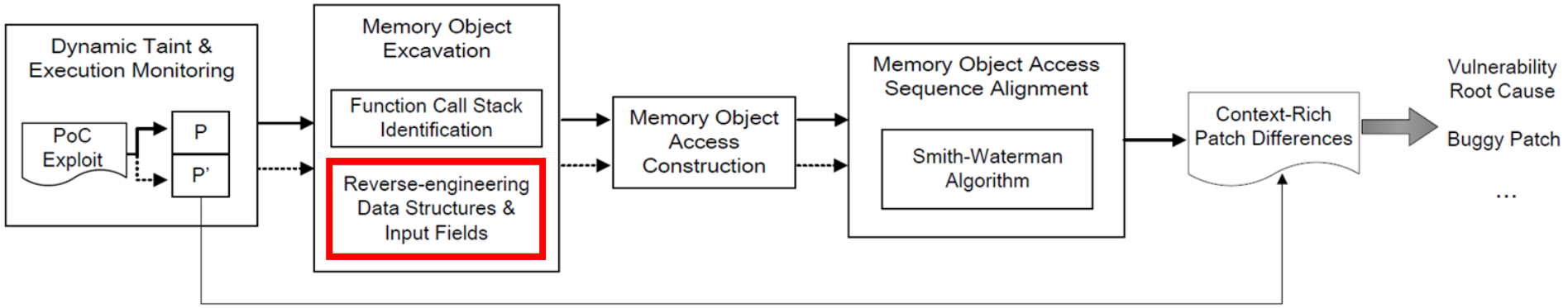


- 第二步：内存对象提取
  - 根指针提取
    - 静态变量：变量的内存地址
    - 栈中的局部变量：所在函数上下文及其相对 ebp 的偏移量
    - 堆中的动态变量： 内存分配函数（如 malloc）的返回值

$$mobj = (alloc, size, type)$$

$$A(mobj) = (mobj, cc, op, optype, \alpha)$$

工作流程

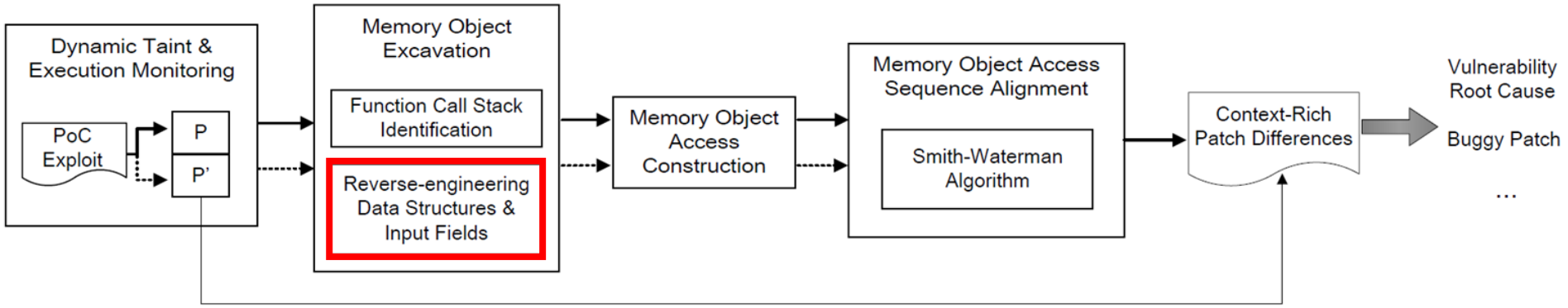


- 第二步：内存对象提取
  - 内存对象大小推断
    - 静态变量/栈中的局部变量：相邻根指针的间隔
    - 堆中的动态变量： 内存分配函数（如 malloc）的参数

mobj = (alloc, size, type)

A(mobj) = (mobj, cc, op, optype, α)

工作流程

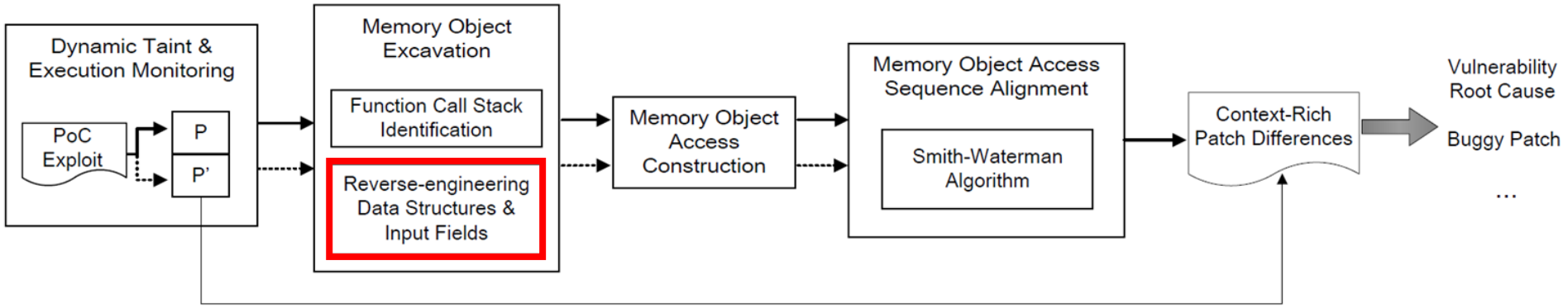


- 第二步：内存对象提取
  - 追踪指针传播
    - 识别根指针及其别名指针
    - 追踪根指针上的算术指令
    - 识别内存访问中内存地址依赖的根指针

mobj = (alloc, size, type)

A(mobj) = (mobj, cc, op, optype, α)

工作流程

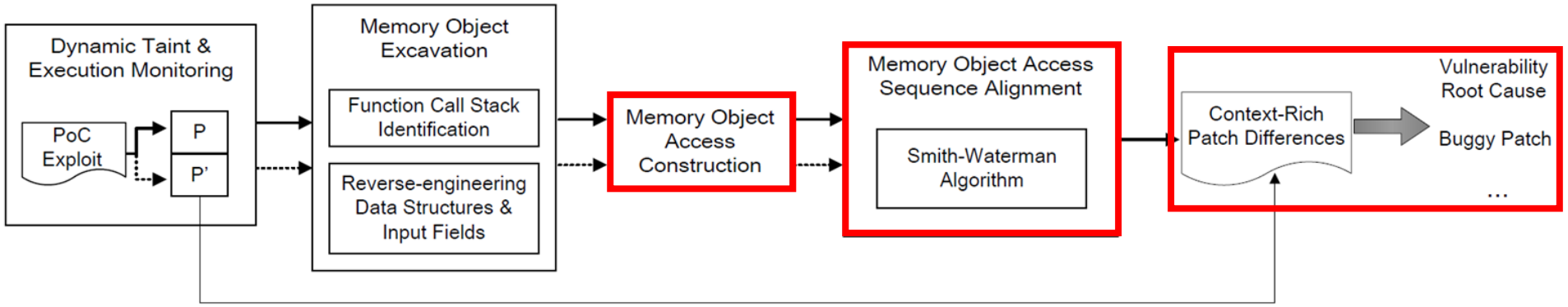


- 第二步：内存对象提取
  - 关联输入字段
    - 识别污点标签
    - 计算污点操作数的地址到根指针的偏移
    - 关联内存对象和输入字段

mobj = (alloc, size, type)

A(mobj) = (mobj, cc, op, optype, α)

工作流程



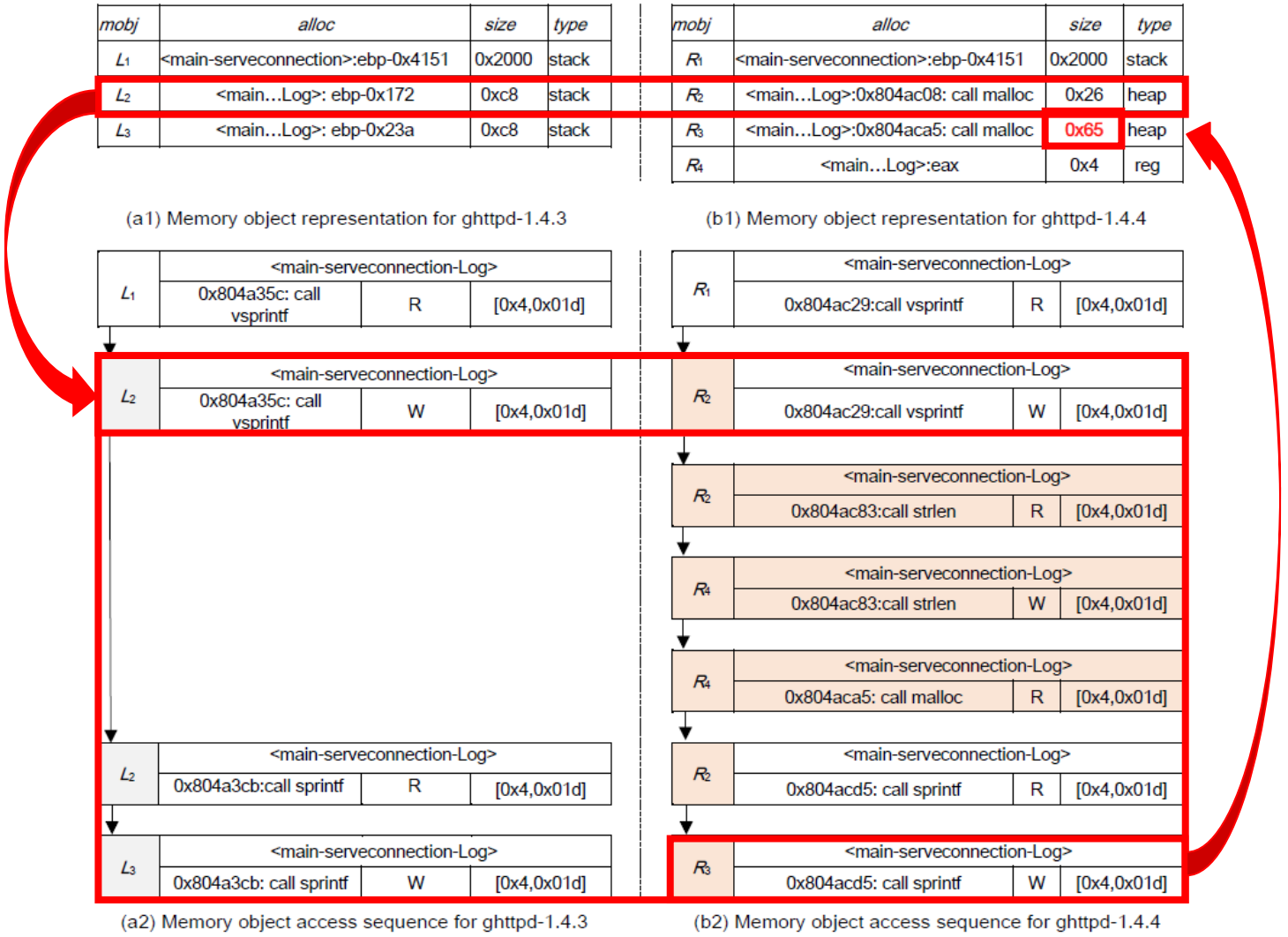
- 第三步：内存对象访问构建
- 第四步：内存对象访问序列匹配
- 第五步：识别补丁差异

mobj = (alloc, size, type)

A(mobj) = (mobj, cc, op, optype, α)

Case study

内存对象
L2与R2不同



内存对象
R3的大小
受输入控制

实验结果分析

- 数据集
  - 可用性
    - PoC 和安全补丁可用
  - 漏洞类型
    - stack overflow, heap overflow, off-by-one, use-after-free, double free
  - 补丁代码模式
    - 覆盖所有补丁代码模式

实验结果分析

- 与现有技术对比
  - 静态分析
    - BinDiff, Diaphora, DarunGrim, DeepBinDiff
    - 设置动态执行轨迹比对
  - 符号执行
    - CoP
  - 语义感知
    - BinSim, BLEX

实验结果分析

- 实验结果
  - 比现有技术更准确
    - 识别出的差异更少
    - 对未引入指令差异的补丁更敏感

Program	Vulnerability		Security Patch		Time(s)		Result (Number of different items detected)											
	CVE	Type	LOC	Type	Trace	Diff	BinDiff (Inst.)		Diaphora (Inst.)		DarunGrim (Basic Block)		DeepBinDiff (Basic Block)		BLEX (Inst.)	CoP (BB.)	BinSim (Syscall)	PatchScope (MOA)
							static	trace	static	trace	static	trace	static	trace				
streamripper-1.61.25	2006-3124	stack OF	6	data struc. & func. PRM	147	123	0	0	0	0	0	0	0	0	361	0	0	1
newspost-2.1	2005-0101	stack OF	2	change checks	141	92	4	2050	8	2052	2	1025	1	1024	2075	2	0	1
mcrypt-2.5.8	2012-4409	stack OF	2	add checks	99	81	15	14	45	38	6	5	4	2	151	3	0	3
tiffsplit-3.8.2	2006-2656	stack OF	5	function calls	108	88	5	5	5	5	3	3	2	2	29	2	4	4
unrar-3.9.3	NA	stack OF	4	function calls	291	135	11	6	11	6	2	1	3	1	41	2	5	3
xmp-2.5.1	2007-6731	stack OF	3	data structures	138	84	1	1	1	1	1	1	1	1	16	1	0	1
gif2png-2.5.2	2009-5018	stack OF	6	add checks	117	104	25	17	131	38	16	11	2	2	203	12	0	5
libsndfile-1.0.25	2015-7805	heap OF	5	function PRM	110	116	33	1579	97	2523	9	850	1	1	94	3	0	3
nasm-0.98.38	2004-1287	stack OF	2	function calls	222	148	3	2	2	2	1	1	4	4	43	1	3	2
0verkill-0.16	2006-2971	numeric error	1	change checks	129	91	2	2	2	2	1	1	0	0	87	1	0	2
ringtonetools-2.22	2004-1292	stack OF	4	data struc. & add checks	179	111	2	4082	2	4082	1	2041	1	2041	2359	1	0	2
Unalz-0.52	2005-3862	stack OF	2	add checks	81	104	22	22	16	16	4	4	2	2	245	3	0	2
O3read-0.03	2004-1288	stack OF	1	add checks	79	99	13	11264	5	5120	4	3072	1	1024	3323	2	0	1
gzip-1.2.4	2001-1228	stack OF	2	function calls	135	151	2	1	2	1	2	1	3	1	601	2	0	2
binutils-2.12	2005-4807	stack OF	8	function calls	103	138	7	2	4	2	4	1	2	1	92	2	2	2
conquest-8.2a	2007-1371	stack OF	5	change checks	1537	189	10	3072	36	5124	11	2049	0	0	1139	8	0	1
poppler-0.24.1	2013-4473	stack OF	4	function calls	444	163	2	2	2	2	1	1	0	0	121	1	0	3
ntpd-4.2.6	2014-9295	stack OF	11	add functions	208	157	52	45	70	55	15	11	5	5	76	7	0	6
libsmi-0.4.8	2010-2891	stack OF	7	change checks	178	125	37	2948	59	3092	8	259	4	258	1112	6	0	5
fontforge-20100501	2010-4259	stack OF	18	function PRM	277	148	0	0	0	0	0	0	0	0	72	0	0	4
2fax-3.04	2004-1255	stack OF	27	add func. & func. PRM	119	135	97	4588	100	5394	12	1080	3	768	1584	14	0	4
libpng-1.2.5	2004-0597	stack OF	25	add checks	69	110	21	8	47	12	13	5	0	0	111	11	0	5
ytree-1.9.4	NA	stack OF	4	function calls	52	82	9	9	9	9	1	1	0	0	81	1	0	3
sox-12.17.4	2004-0557	stack OF	8	add checks	101	90	76	27	26	13	8	3	6	3	372	5	0	4
ettercap-0.7.5.1	2013-0722	stack OF	2	function PRM	405	106	0	0	0	0	0	0	0	0	128	0	0	1
binutils-2.15	2006-2362	stack OF	15	add checks	157	143	107	32	189	28	35	10	8	2	337	25	0	5
prozilla-1.3.6	2004-1120	stack OF	10	function calls	189	130	11	2	21	6	6	1	6	1	436	6	0	3
nginx-1.4.0	2013-2028	numeric error	4	add checks	88	97	49	12	9	4	3	2	3	2	210	3	0	6
proftpd-1.3.3a	2010-4221	stack OF	4	add checks	111	91	5	5	3	3	1	1	1	1	189	1	0	2
tiff2pdf-4.0.9	2018-15209	heap OF	8	add checks	116	91	111	47	202	52	40	8	10	4	527	28	0	5
nginx-1.12.0	2017-7529	numeric error	3	add checks	309	173	15	8	16	10	6	2	3	1	532	4	0	3
Aircrack-ng-1.2	2014-8322	stack OF	2	add checks	455	187	6	3	34	2	2	1	2	1	324	2	0	3
leptonica-1.70.1	2018-7186	stack OF	2	function PRM	97	131	0	0	0	0	0	0	0	0	319	0	0	1
openjpeg-2.1.1	2016-7445	Null pointer	4	add checks	163	149	48	11	24	3	9	2	10	8	362	8	0	4
Jasper-1.900.9	2016-8887	Null pointer	5	function calls	193	127	32	23	24	15	5	4	4	4	144	4	0	3
libzip-1.2.0	2017-12858	double free	3	function calls	125	113	40	16	108	15	12	6	8	5	514	8	1	1
GraphicsMagick-1.3.26	2017-14103	UAF	51	function calls	643	190	275	35	361	49	50	10	19	10	634	41	11	5
putty-0.66	2016-2563	stack OF	NA	NA	339	176	675	63	696	21	110	5	33	9	4564	87	18	12
mutt-1.4.2.2	2007-2683	stack OF	NA	NA	312	131	31	2	33	4	8	1	9	2	148	4	6	2
inetutils-1.8	2011-4862	stack OF	NA	NA	137	159	2483	436	2506	597	441	155	207	81	6029	135	41	17
Apache-1.3.35	2006-3747	off-by-one	NA	NA	358	168	198	0	198	0	21	0	45	0	158	16	38	0
xrdp-0.4.1	2008-5904	heap OF	NA	NA	284	141	7	7	9	8	2	2	2	2	342	2	2	5
alsaplayer-0.99.80-rc2	2007-5301	stack OF	NA	NA	252	111	16	2	16	2	1	1	0	0	147	1	0	2
opendchub	2010-1147	stack OF	NA	NA	203	126	56	8	73	25	17	4	27	9	266	14	18	2
nasm-2.14	2018-19214	heap OF	NA	NA	209	125	1144	394	1409	493	166	46	33	24	3672	63	27	13
In Total					**10209**	**5729**	**5758**	**30852**	**6611**	**28926**	**1060**	**10687**	**475**	**5306**	**34370**	**542**	**176**	**164**

实验结果分析

- 误报率和漏报率
- 细粒度的补丁差异

		FPR	FNR
BinDiff	static	42.23%	15.56% (7/45)
	trace	32.67%	15.56% (7/45)
Diaphora	static	39.29%	15.56% (7/45)
	trace	29.37%	15.56% (7/45)
DarunGrim	static	33.75%	15.56% (7/45)
	trace	22.76%	15.56% (7/45)
DeepBinDiff	static	18.31%	26.67% (12/45)
	trace	12.12%	26.67% (12/45)
BLEX		81.56%	2.22% (1/45)
CoP		27.12%	51.11% (23/45)
BinSim		0.00%	73.33% (33/45)
PATCHSCOPE		14.73%	2.22% (1/45)

		Impacts on input manipulations		
		cut the inputs	filter the inputs	None
Diff in MO	*alloc*	0	0	1
	size	3	0	0
	type	1	0	0
Diff in MOA	*cc*	0	17	0
	op	7	0	5
	α	10	0	0

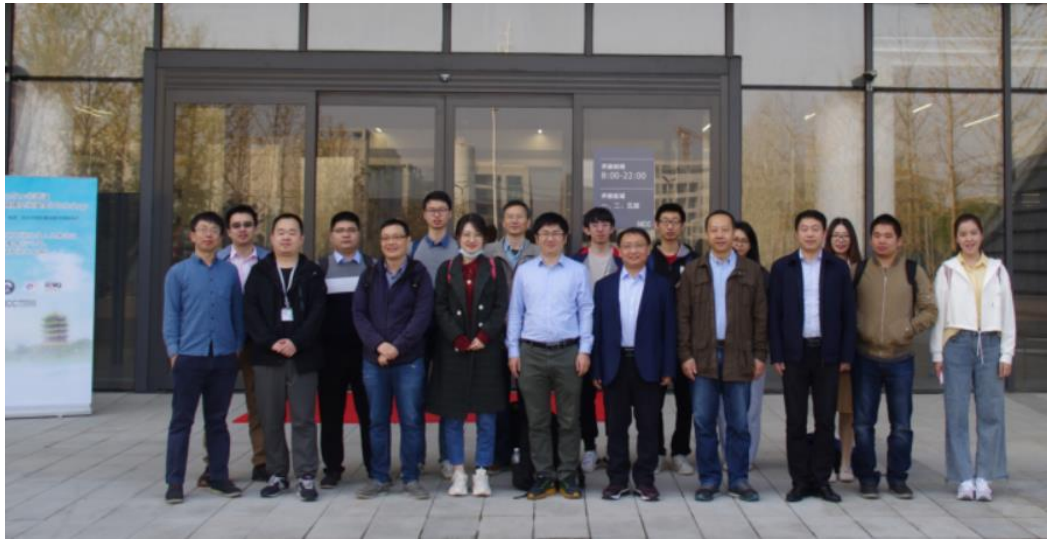
讨论

- 可扩展性问题
  - 在二进制层面识别补丁模式
- 欠污染问题
  - 追踪隐式数据依赖
- Patch 的漏洞类型
  - 主要针对的是内存损坏性漏洞
  - 理论上对输入相关的权限绕过漏洞也有效
- 其他可扩展的应用场景
  - Carsh 分析、指导补丁生成……

总结

- 对补丁代码模式进行了大规模实证研究
  - 补丁代码模式对代码改动的影响
  - 二进制层面定位补丁差异面临的挑战
- 基于内存对象访问序列的二进制补丁比对技术
  - 对不同类型的补丁具有鲁棒性
  - 揭示了丰富的语义信息
    - 程序在动态执行中如何通过访问各种内存对象来操作输入字段
  - 更加简洁和准确的分析结果

国家网络安全人才与创新基地



Q&A

欢迎交流：xkernel@whu.edu.cn
leizhao@whu.edu.cn