

从代码安全分析 走向 代码安全治理

蜚语科技 / 杨文博

目录

contents

0

关于我们

1

02

代码分析与应用现代化

0

从代码安全分析走向代码安全治理

3

0

大语言模型与代码安全治理

4

01

关于我们

简介

杨文博

- 蜚语科技创始人&CTO
- 博士毕业于上海交通大学计算机系
- 多年软件安全、漏洞攻防、程序分析研究经验
- 0ops前CTF选手
- 多篇顶会论文
- 多个授权发明专利
- 多次行业会议技术分享
- 上海市科技进步一等奖
- 参与多项国家科研、产业技术项目



02

代码分析 与 应用现代化

解决什么问题

帮助程序员**自动**的分析代码，
寻找代码中的**潜在问题(Bug与漏洞)**

为什么要做这些

全球每年产生~93,000,000,000行代码

每千行代码(KLOC), 包含10-20个严重BUG, 7-12个安全漏洞

平均修复一个严重BUG/漏洞的时间是
0.75天/205天

最终目标

帮助程序员自动的分析代码，
寻找代码中的潜在问题(Bug与漏洞)
以及，做一些更多的事



节省时间投入，提升研发效率

代码分析技术与工具 - 代码风格

➤ include文件的命名与顺序

include文件必须遵循以下的顺序：相关头文件、C系统头文件、C++基础库头文件、其他库头文件、本项目头文件

```
01. #include "foo/server/fooserver.h"
02.
03. #include <sys/types.h>
04. #include <unistd.h>
05.
06. #include <string>
07. #include <vector>
08.
09. #include "base/basictypes.h"
10. #include "foo/server/bar.h"
11. #include "third_party/absl/flags/flag.h"
```

➤ 函数声明与定义

返回值的类型必须与函数名在同一行
参数尽量与函数名在同一行
函数名称与开括号之间永远没有空格。
如果可能，所有参数都应对齐。
默认缩进为2个空格。

```
01. class Shape {
02. public:
03.     virtual void Rotate(double radians) = 0;
04. };
05.
06. class Circle : public Shape {
07. public:
08.     void Rotate(double radians) override;
09. };
10.
11. void Circle::Rotate(double /*radians*/) {}
```

➤ 每一行代码长度

每一行代码不能超过80个字符

```
01. // Iterates over the contents of a GargantuanTable.
02. // Example:
03. // std::unique_ptr<GargantuanTableIterator> iter = table->NewIterator();
04. // for (iter->Seek("foo"); !iter->done(); iter->Next()) {
05. //     process(iter->key(), iter->value());
06. // }
07. class GargantuanTableIterator {
08. ...
09. };
```

➤ 只使用space而不使用tab

➤ 使用一致的注释风格

➤ 等等

代码分析技术与工具 - 代码风格

技术特点

- 相关结果主要影响代码的可维护性、一致性
- 相关代码结构较为简单，一般不涉及上下文
- 定制化程度较高，不同团队的需求不尽相同
- 对代码分析能力要求不高
- 核心技术为模式匹配

工具化

- 以开源工具为主，比如sonarqube、cpplint等
- 研发团队基于开源工具进行定制
- 准确性较高，适合作为卡点

代码分析技术与工具- 代码质量/功能安全

- misra-c2012-R.15.5 函数最后应该有一个退出点
- misra-c2012-R.22.1 所有通过标准库函数动态获取的资源都应显式释放
- GJB8114-2013-R-1-3-1 禁止指针的指针超过两级
- GJB8114-2013-R-1-6-8 数组禁止越界使用
- GJB8114-2013-R-1-6-15 缓存区写入操作禁止越界
- 等等

```
bool_t f ( uint16_t n, char *p )
{
    if ( n > MAX )
    {
        return false;
    }

    if ( p == NULL )
    {
        return false;
    }

    return true;
}
```

```
void comp(int a[], int n) {
    int i;
    for (i = 0; i <= n; i++) { // 违背1
        a[i] = 0;
    }
}

int test1(void) {
    int array[100];
    comp(array, 100);
    array[100] = 1; // 违背2
    return (0);
}
```

```
#include <string.h>

int test1(void) {
    int src[4] = {1, 2, 3, 4};
    int des[2] = {0, 0};
    memcpy(des, src, sizeof(src)); // 违背1
    return (0);
}
```

代码分析技术与工具-代码质量/功能安全

技术特点

- 相关结果主要影响代码的可维护性、一致性、稳定性、性能
- 相关代码结构不复杂，涉及一定的上下文信息，部分涉及程序动态数据
- 对代码分析能力有一定要求，整体误报率/漏报率易于控制
- 以C/C++为主，存在成熟的标准(如MISRA、AUTOSAR、GJB、DO-178B/C等等)
- 核心技术为模式匹配、数据流分析、控制流分析

工具化

- 存在开源工具，比如sonarqube
- 商业工具对标准的支持更全面，完备性更高，比如Coverity、Klocwork、QAC、Corax等
- 存在国产化替代产品
- 大部分规则准确性较高，适合作为卡点

代码分析技术与工具 - 代码安全

- 不要读取未初始化内存
- 不要对空指针解引用
- 不充分的随机数
- 整型溢出
- SQL注入
- 路径遍历
- 反序列化
- 硬编码密钥
- 等等

```
void set_flag(int number, int *sign_flag) {  
    if (NULL == sign_flag) {  
        return;  
    }  
  
    if (number > 0) {  
        *sign_flag = 1;  
    } else if (number < 0) {  
        *sign_flag = -1;  
    }  
}  
  
int is_negative(int number) {  
    int sign;  
    set_flag(number, &sign);  
    return sign < 0;  
}
```

```
#include <png.h> /* 源于 libpng */  
#include <string.h>  
void func(png_structp png_ptr, int length, const void *user_data) {  
    png_charp chunkdata;  
    chunkdata = (png_charp)png_malloc(png_ptr, length + 1);  
    /* ... */  
    memcpy(chunkdata, user_data, length);  
    /* ... */  
}
```

```
Session session = sessionFactory.openSession();  
Query q = session.createQuery("select t from UserEntity t where id = " + input);  
q.execute();
```

代码分析技术与工具-代码安全

技术特点

- 相关结果主要影响代码的稳定性、安全性
- 相关代码结构复杂，涉及大量上下文信息与程序动态数据
- 对代码分析能力很高要求，整体误报率/漏报率难以控制
- C/C++与其他语言存在差异，代码安全也影响稳定性
- 部分语言存在成熟的标准(如CERT)，更多参考CWE与OWASP分类
- 核心技术扩展至部分形式化技术(模型检查、符号执行、抽象解释)及人工智能技术(自然语言处理)

代码分析技术与工具-代码安全

工具化

- 大量开源工具，比如CodeQL、Infer、CSA、PMD、Semgrep、CBMC
- 商业工具的分析能力更强，比如Coverity、Fortify、QAC、Corax等，拥有更好的产品化能力、漏报率与误报率、兼容性、性能。
- 现有工具仍然存在缺陷，距离理想目标尚远
- 国产化产品距离世界一流水平尚有差距

现代研发体系下代码分析的困境与挑战

- 分析场景日益复杂，分析能力不足，结果准确性不高
- 止步于给出分析结果，对于问题的治理难以闭环

突破传统程序分析技术上限

NP困难: 在静态下判定一个程序所有的运行状态

在工程实践中, 很多复杂的代码问题在静态状态不可判定

用工程的方法, 在有限的算力/时间下, 逼近理论的极限

对程序的运行状态进行剪枝、舍弃

代码覆盖率下降, 出现漏报



NLP+程序分析

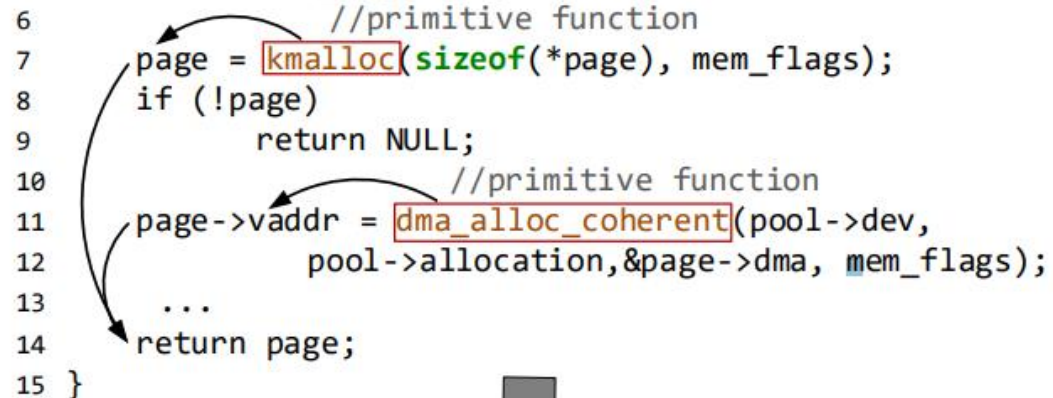
```
1 /*File: Linux/drivers/video/fbdev/hyperv_fb.c*/
2 static int hvfb_probe(struct hv_device *hdev,
3                       const struct hv_vmbus_device_id *dev_id)
4 {
5     struct fb_info *info;
6     struct hvfb_par *par;
7     int ret;
8     ...
9     /*ALLOCATE OBJECT info*/
10    ❶ info = framebuffer_alloc(sizeof(struct hvfb_par),
11                              &hdev->device);
12    if (!info)
13        return -ENOMEM;
14    ...
15    ret = hvfb_getmem(hdev, info);
16
17    if (ret) {
18        pr_err("No memory for framebuffer\n");
19        goto error2;
20    }
21    ...
22 error2:
23    ...
24    /*DEALLOCATE OBJECT info*/
25    ❷ framebuffer_release(info);
26    ... //info->apertures double free!
27    return ret;
28 }
```

```
29 struct fb_info *framebuffer_alloc(size_t size, ...)
30 {
31     struct fb_info *info;
32     char *p;
33     ...
34     p = kzalloc(fb_info_size + size, GFP_KERNEL);
35     info = (struct fb_info *) p;
36     ...
37     return info;
38 }
39 static int hvfb_getmem(struct hv_device *hdev, ...)
40 {
41     ...
42    ❸ info->apertures = alloc_apertures(1);
43     ... //info->apertures allocation
44     pdev = pci_get_device(...);
45     if (!pdev) {
46         ❹ kfree(info->apertures); //info->apertures free
47         return -ENODEV;
48     }
49 }
50 void framebuffer_release(struct fb_info *info)
51 {
52     if (!info) return;
53    ❺ kfree(info->apertures);
54     kfree(info);
55 }
```

用NLP+程序分析技术，在可控的算力/
时间成本下，找回丢失的代码覆盖率

NLP+程序分析

```
1 /*File: Linux/mm/dmapool.c*/
2 static struct dma_page *pool_alloc_page(struct
3     dma_pool *pool, gfp_t mem_flags)
4 {
5     struct dma_page *page;
6     //primitive function
7     page = kmalloc(sizeof(*page), mem_flags);
8     if (!page)
9         return NULL;
10    //primitive function
11    page->vaddr = dma_alloc_coherent(pool->dev,
12        pool->allocation, &page->dma, mem_flags);
13    ...
14    return page;
15 }
```



MOS Representation:

Function name	Property
pool_alloc_page	: Allocator,
Memory object list	Object type
RetVal	: struct dma_page*,
RetVal->vaddr	: void*

ID	Program	File:line_number	Buggy Function	MM Function	Type	Confirmed
1	Linux	drivers/gpu/drm/xen/xen_drm_front.c:555	xen_drm_drv_init	kfree	UAF	✓
2	Linux	drivers/infiniband/sw/siw/siw_mem.c:116	siw_alloc_mr	siw_mem_put	UAF	✓
3	Linux	drivers/scsi/myrs.c:2281	myrs_cleanup	iounmap	DF	✓
4	Linux	drivers/ipack/carriers/tpci200.c:600	tpci200_pci_probe	tpci200_uninstall	DF	✓
5	Linux	drivers/net/ethernet/qlogic/qlcnict/qlcnict_minidump.c:1424	qlcnict_83xx_get_minidump_template	vfree	UAF	✓
6	Linux	lib/test_kmod.c:1148	register_test_dev_kmod	vfree	UAF	
7	Linux	drivers/dma/dmaengine.c:1088	dma_async_device_register	free_percpu	DF	✓
8	Linux	drivers/net/ethernet/myricom/myri10ge/myri10ge.c:2897	myri10ge_sw_tso	dev_kfree_skb_any	UAF	✓
9	Linux	drivers/net/ethernet/netronome/nfp/bpf/cmsg.c:456	nfp_bpf_ctrl_msg_rx	dev_kfree_skb_any	UAF	✓
10	Linux	drivers/net/wireless/intersil/hostap/hostap_80211_rx.c:1019	hostap_80211_rx	dev_kfree_skb_any	UAF	
11	Linux	drivers/net/wireless/marvell/mwifiex/tlds.c:859	mwifiex_send_tlds_action_frame	dev_kfree_skb_any	DF	✓
12	Linux	drivers/net/wireless/ath/ath10k/htc.c:656	ath10k_htc_send_bundle	dev_kfree_skb_any	UAF	✓
13	Linux	drivers/crypto/qat/qat_common/adf_transport.c:173	adf_create_ring	dma_free_coherent	DF	✓
14	Linux	drivers/net/ethernet/broadcom/bcm4908_enet.c:174	bcm4908_enet_dma_alloc	dma_free_coherent	DF	✓
15	Linux	drivers/net/wireless/marvell/mwl8k.c:1474	mwl8k_probe_hw	dma_free_coherent	DF	✓
16	Linux	drivers/net/wireless/intel/iwlwifi/queue/tx.c:1101	iwl_txq_dyn_alloc_dma	dma_free_coherent	DF	
17	Linux	drivers/mtd/nand/raw/gpmi-nand/gpmi-nand.c:2477	gpmi_nand_init	dma_free_coherent	DF	✓
18	Linux	drivers/scsi/be2iscsi/be_mgmt.c:533	beiscsi_if_clr_ip	dma_free_coherent	UAF	
19	Linux	drivers/gpu/drm/i915/gt/gen8_ppgtt.c:631	gen8_preallocate_top_level_pdp	i915_gem_object_put	UAF	✓
20	Linux	drivers/misc/ibmasm/remote.c:265	ibmasm_init_one	input_free_device	UAF	✓
21	Linux	drivers/misc/ibmasm/remote.c:266	ibmasm_init_one	input_free_device	UAF	✓
22	Linux	net/tipc/socket.c:1268	tipc_sk_mcast_rcv	kfree_skb	DF	✓
23	Linux	drivers/net/ethernet/qualcomm/emac/emac-mac.c:1459	emac_mac_tx_buf_send	kfree_skb	UAF	✓
24	Linux	drivers/net/ethernet/cisco/enic/enic_main.c:860	enic_hard_start_xmit	kfree_skb	UAF	✓
25	Linux	net/ipv6/ip6_tunnel.c:1439	ip6_tnl_start_xmit	kfree_skb	DF	
26	Linux	drivers/scsi/bnx2fc/bnx2fc_fcoe.c:444	bnx2fc_rcv	kfree_skb	DF	
27	Linux	net/nfc/digital_dep.c:1287	digital_tg_recv_dep_req	kfree_skb	DF	✓
28	Linux	drivers/video/fbdev/hyperv_fb.c:1273	hvfb_probe	framebuffer_release	DF	✓
29	Linux	drivers/net/wan/hdlc_fr.c:417	pvc_xmit	__skb_pad	DF	✓
30	Linux	drivers/firmware/efi/efi.c:937	efi_mem_reserve_persistent	memunmap	UAF	✓
31	Linux	net/rds/message.c:350	rds_message_map_pages	rds_message_put	UAF	✓
32	Linux	drivers/infiniband/ulp/iser/ib_isert.c:477	isert_connect_request	isert_device_put	UAF	✓

在多个知名开源项目中，发现上百个全新的bug

我们想要去到的地方

- 分析场景日益复杂，分析能力不足，结果准确性不高
- 止步于给出分析结果，对于问题的治理难以闭环

03

从代码安全分析 走向 代码安全治理

代码安全分析与代码安全治理

Coding



研发

触发扫描



Jenkins



判定结果



研发/安全

修复代码



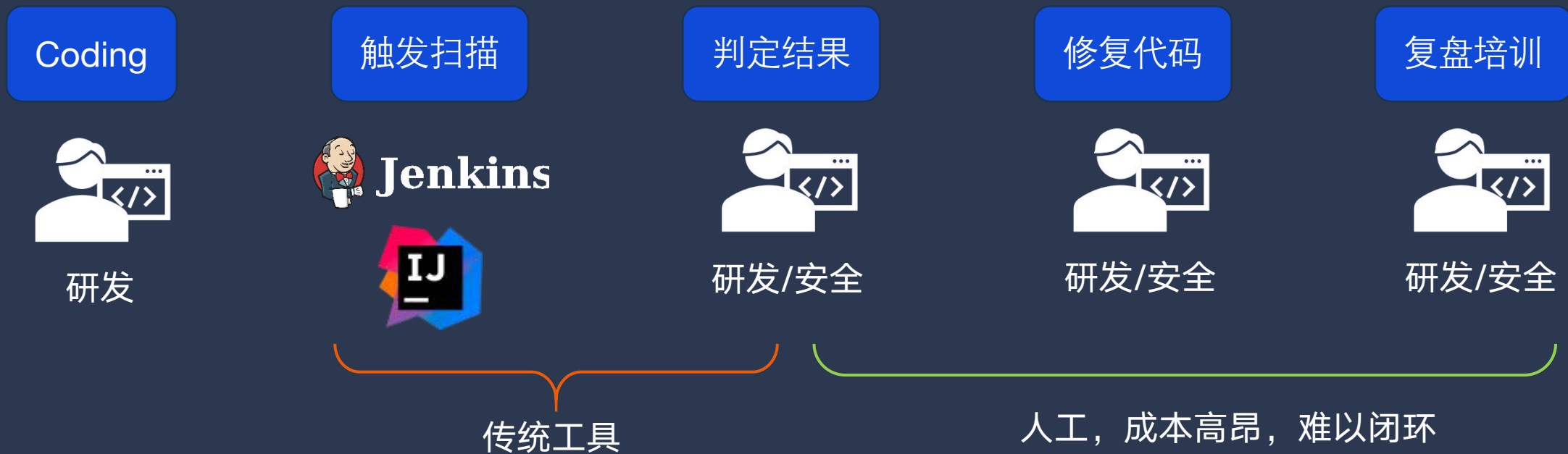
研发/安全

复盘培训



研发/安全

代码安全治理的“最后一公里”



分析结果距离PoC 有着很长的距离

File Edit Tools Options Help

Filter Set: Quick View My Issues

18 11 0 0 ... 29

Critical (18)

Group By: Category

Command Injection - [0 / 1]

LogServiceImpl.java:108 (Command Injection)

Dynamic Code Evaluation: JNDI Reference Injection - [0 / 1]

LogController.java:39 (Dynamic Code Evaluation: JNDI Reference Injection)

Open Redirect - [0 / 1]

UserController.java:48 (Open Redirect)

Password Management: Hardcoded Password - [0 / 1]

OrderFilter.java:94 (Password Management: Hardcoded Password)

Path Manipulation - [0 / 7]

FileUtil.java:39 (Path Manipulation)

FileUtil.java:48 (Path Manipulation)

FileUtil.java:88 (Path Manipulation)

UserServiceImpl.java:207 (Path Manipulation)

UserServiceImpl.java:221 (Path Manipulation)

ZipUtils.java:14 (Path Manipulation)

ZipUtils.java:18 (Path Manipulation)

Privacy Violation - [2 / 5]

DecryptionRequestUtil.java:70 (Privacy Violation)

SignUtils.java:33 (Shared Sink) - [2 / 2]

SignUtils.java:58 (Privacy Violation)

SM2Util.java:118 (Privacy Violation)

Weak Encryption: Insecure Mode of Operation - [0 / 2]

AesUtil.java:31 (Weak Encryption: Insecure Mode of Operation)

AesUtil.java:53 (Weak Encryption: Insecure Mode of Operation)

BenchmarkTest02104.java OrderFilter.java FileUtil.java UserServiceImpl.java LogServiceImpl.java LogController.java 33 Project Summary

```
1 package org.itstec.log.controller;
2
3 import org.apache.logging.log4j.LogManager;
4
5
6
7
8
9
10
11
12
13 @RestController
14 @RequestMapping("/log")
15 public class LogController {
16
17     private static Logger log= LogManager.getLogger(LogController.class);
18
19     @Autowired
20     private LogService logService;
21
22     @RequestMapping(value = "/show",method = {RequestMethod.POST})
23     public R<> show(String logPath){
24         return logService.show(logPath);
25     }
26
27     @RequestMapping(value = "/logGrab",method = {RequestMethod.POST})
28     public R<> logGrab(String logPath){
29         return logService.logGrab(logPath);
30     }
31
32     @RequestMapping(value = "/logQuickArch",method = {RequestMethod.POST})
33     public R<> logQuickArch(){
34         return logService.logQuickArch();
35     }
36
37     @RequestMapping(value = "/logArch",method = {RequestMethod.POST})
38     public R<> logArch(String para){
39         log.info("logArch:{}",para);
40         return logService.logArch(para);
41     }
42 }
43
44
```

Analysis Trace 33

Abstract:

程序使用 LogController.java 中的第 39 行不受信任的地址运行 JNDI 查找，这可能使攻击者能够远程运行任意 Java 代码。

Explanation:

如果攻击者可以控制 JNDI 查找操作的地址，则他通过将该地址指向其控制的服务器的地址并将 JNDI 命名引用返回到具有自定义对象工厂的 RMI 存储对象，也许能够远程运行任意代码。

在这种情况下，JNDI 查找操作在 LogController.java 的第 39 行执行。

示例 1：以下代码会使用不可信数据运行 JNDI 查找。

...

String address = request.getParameter("address");

Properties props = new Properties();

props.put(Provider_URL, "rmi://secure-server:1099/");

InitialContext ctx = new InitialContext(props);

ctx.lookup(address);

使用 Apache Log4j 2.14.1 及更早版本的应用程序可能容易受到 CVE-2021-44228 的攻击。当使用易受攻击的 log4j 版本记录用户输入时，攻击者可能会发送要记录的恶意有效负载，以触发远程服务器上的 JNDI 查找，从而导致执行任意代码。

示例 2：以下代码显示了易受攻击的 log4j 记录器实例如何受到攻击：

更丰富的控制流/数据流信息

高 [0110] SQL注入-C:\Users\notify\Desktop\java\test\altorj\Altorj\src\com\ib...

文件 C:\Users\notify\Desktop\java\test\altorj\Altorj\src\com\ibm\security\appscan\altoromutual\...

路径提示

未审核

267 * @return

268 * @throws SQLException

269 */

270 public static Account[] getAccounts(String username) throws SQLException{

271 if (username == null || username.trim().length() == 0)

272 return null;

273

274 Connection connection = getConnection();

275 Statement statement = connection.createStatement();

276

ResultSet resultSet =statement.executeQuery("SELECT ACCOUNT_ID, ACCOUNT_NAME, BALANCE FROM ACCOUNTS WHERE USERID = '"+ username +"

12 < [3] 关键路径: \$stack17 = virtualinvoke \$stack16.<java.lang.StringBuilder: java.lang.StringBuilder append(java.lang.String)>(username) →

13 < [3] 关键路径: \$stack18 = virtualinvoke \$stack17.<java.lang.StringBuilder: java.lang.StringBuilder append(java.lang.String)>("\'") →

14 < [3] 关键路径: \$stack19 = virtualinvoke \$stack18.<java.lang.StringBuilder: java.lang.String toString>() →

15 < [3] 关键路径: \$stack20 = interfaceinvoke \$stack14.<java.sql.Statement: java.sql.ResultSet executeQuery(java.lang.String)>(\$stack19) →

16 < 使用 '<java.sql.Statement: java.sql.ResultSet executeQuery(java.lang.String)>' 可能容易受到 SQL 注入的攻击

277

278 ArrayList<Account> accounts = new ArrayList<Account>(3);

279 while (resultSet.next()){

280 long accountId = resultSet.getLong("ACCOUNT_ID");

281 String name = resultSet.getString("ACCOUNT_NAME");

282 double balance = resultSet.getDouble("BALANCE");

283 Account newAccount = new Account(accountId, name, balance);

284 accounts.add(newAccount);

缺陷信息

路径 (2)

批注

修复建议

▶ 触发路径1 (深度2)

▼ 触发路径2 (深度16)

1 LoginServlet.java:75-[0] 关键路径: \$stack7 = interfaceinvoke request

2 LoginServlet.java:77-[0] 关键路径: \$stack26 = virtualinvoke \$stack7.

3 LoginServlet.java:77-[0] 关键路径: username#2 = virtualinvoke \$sta

4 LoginServlet.java:80-[0] 关键路径: \$stack10 = virtualinvoke \$stack9

5 LoginServlet.java:82-[0] >>: \$stack11 = staticinvoke <com.ibm.secu

6 DBUtil.java:219-[1] 关键路径: \$stack19 = interfaceinvoke \$stack11.<

7 LoginServlet.java:94-[0] >>: \$stack12 = staticinvoke <com.ibm.secu

8 ServletUtil.java:342-[1] >>: \$stack6 = staticinvoke <com.ibm.securit

9 DBUtil.java:259-[2] 关键路径: virtualinvoke \$stack19.<com.ibm.secu

10 ServletUtil.java:343-[1] >>: \$stack7 = virtualinvoke \$stack6.<com.it

11 User.java:78-[2] >>: \$stack3 = staticinvoke <com.ibm.security.appsp

12 DBUtil.java:276-[3] 关键路径: \$stack17 = virtualinvoke \$stack16.<ja

13 DBUtil.java:276-[3] 关键路径: \$stack18 = virtualinvoke \$stack17.<ja

14 DBUtil.java:276-[3] 关键路径: \$stack19 = virtualinvoke \$stack18.<ja

15 DBUtil.java:276-[3] 关键路径: \$stack20 = interfaceinvoke \$stack14.<

16 DBUtil.java:276-使用 '<java.sql.Statement: java.sql.ResultSet execu

基于程序分析技术的漏洞自动化修复

基于强大的底层分析能力，为部分类型的漏洞生成自动化修复代码

```
void foo()
{
    FILE * p = fopen("test.bin", "rb");
    ...
}
```

```
void foo()
{
    int * p = new int;
    ...
    delete p;
    ...
    f(p);
}
```

```
int foo()
{
    strncat(dest, "*****", sizeof(dest));
    ...
    strcpy(dest, NULL);
    ...
    return strlen(0);
}
```

```
void foo()
{
    FILE * p = fopen("test.bin", "rb");
    ...
    fclose(p); /* [Auto Fix] */
}
```

```
void foo()
{
    int * p = new int;
    ...
    delete p;
    p = nullptr; /* [Auto Fix] */
    ...
    if (!p) f(p); /* [Auto Fix] */
}
```

```
int foo()
{
    strncat(dest, "*****", sizeof(dest) - strlen(dest) - 1); /* [Auto Fix] */
    ...
    (NULL ? strcpy(dest, NULL) : 0); /* [Auto Fix] */
    ...
    return (0 ? strlen(0) : 0); /* [Auto Fix] */
}
```

我们在前进
但是还不够

传统的技术存在局限性

我们需要拥抱新的技术



04

大语言模型 与 代码安全治理

大模型能做什么？
大模型适合做什么？
大模型带来了什么样的变化？

LLM，能够用来做代码分析吗？能



请帮我找出以下代码中的bug：

```
int testARR30()
{
    int x = 9936;
    int j;
    int t[9936] = {0};
    int i;
    for(i = 0; i <= x; i++)
    {
        j = t[i];
    }
    return j;
}
```



这段代码存在数组越界的漏洞。



具体来说，数组 `t` 的大小为 9936，但是在循环中使用 `i <= x` 的条件，当 `i` 的值等于 9936 时，会访问 `t` 数组的第 9937 个元素，从而导致数组越界。这可能会导致程序崩溃或者产生不可预期的结果。

为了避免这个问题，应该将循环的条件改为 `i < x`。

LLM，适合用来做静态代码分析吗？



请帮我找出以下代码中的bug:

```
int testARR30()
{
    int x = 9936;
    int j;
    int t[9936] = {0};
    int i;
    for(i = 0; i <= x; i++)
    {
        j = t[i];
    }
    return j;
}
```



这段代码存在数组越界的漏洞。



具体来说, 数组 `t` 的大小为 9936, 但是在循环中使用 `i <= x` 的条件, 当 `i` 的值等于 9936 时, 会访问 `t` 数组的第 9937 个元素, 从而导致数组越界。这可能会导致程序崩溃或者产生不可预期的结果。

为了避免这个问题, 应该将循环的条件改为 `i < x`。



请帮我找出以下代码中的bug:

```
int testARR30()
{
    int x = 9935;
    int j;
    int t[9936] = {0};
    int i;
    for(i = 0; i <= x; i++)
    {
        j = t[i];
    }
    return j;
}
```



这段代码存在一个越界访问的漏洞。



具体来说, 数组 `t` 的大小为 9936, 但是在循环中使用 `i <= x` 的条件, 当 `i` 的值等于 9935 时, 会访问 `t` 数组的第 9936 个元素。虽然数组 `t` 的大小足够大, 但是当访问超出数组范围时, 可能会访问到未定义的内存位置, 从而导致程序出现未定义行为, 例如崩溃或者产生奇怪的结果。

为了避免这个问题, 应该将循环的条件改为 `i < x`, 以确保不会访问数组范围之外的内存位置。

// Simple ? condition that assigns constant to bar on true condition↓

int num = 106;↓

bar = (7 * 18) + num > 200 ? "This_should_always_happen" : param;↓

String fileName = null;↓

java.io.FileInputStream fis = null;↓

try {↓

fileName = org.owasp.benchmark.helpers.Utils.TESTFILES_DIR + bar;↓

fis = new java.io.FileInputStream(new java.io.File(fileName));↓



这段代码存在一个潜在的安全漏洞，即路径遍历攻击（也称为目录遍历攻击）。攻击者可能会利用这个漏洞来访问、读取或修改您系统中不应该暴露的敏感文件。

问题出在这一行代码上：

java

Copy code

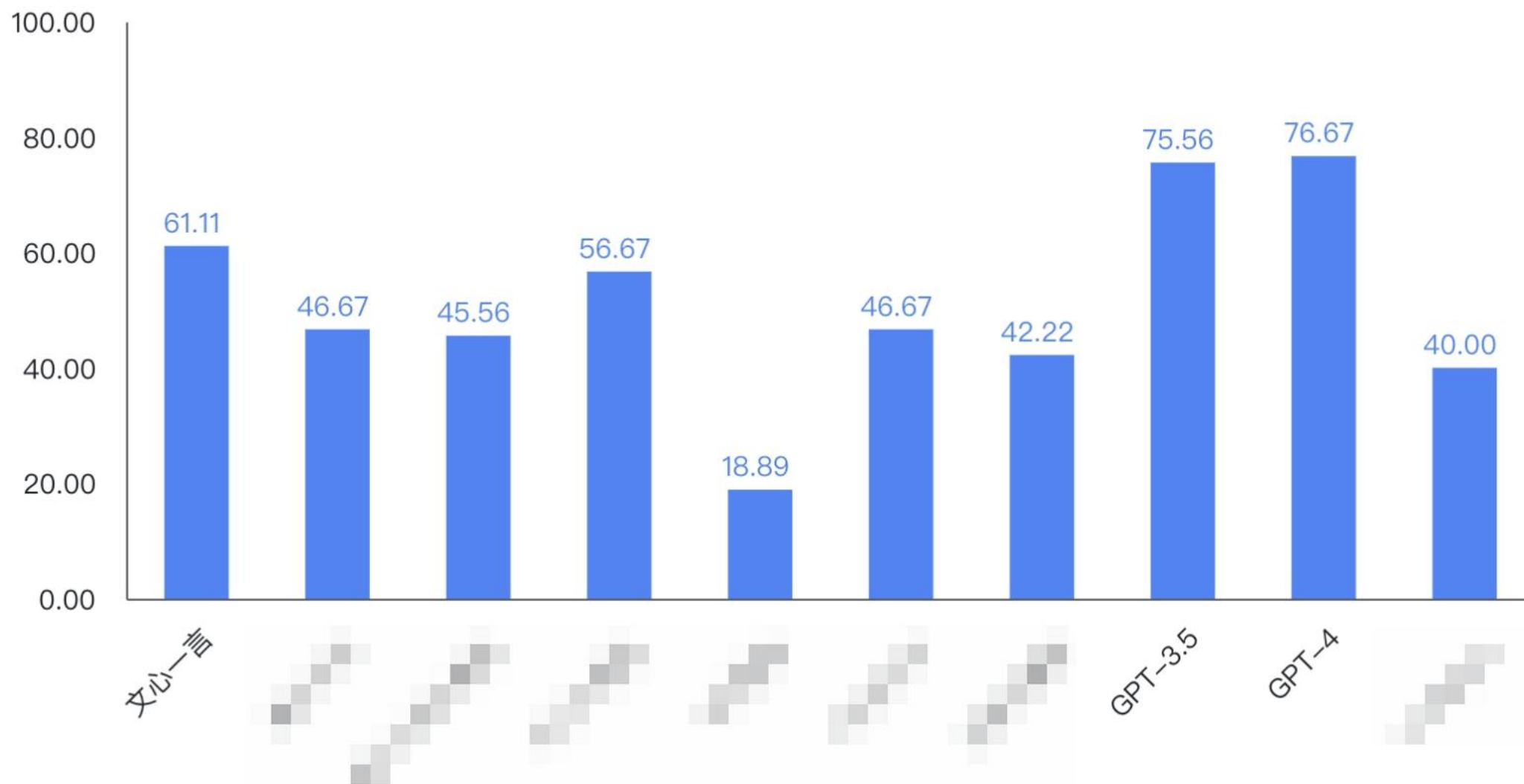
```
fileName = org.owasp.benchmark.helpers.Utils.TESTFILES_DIR + bar;
```

变量`bar`的值可能是用户可控的，因为它的值可能受到`num`和`param`变量的影响。攻击者可以构造一个特殊的`param`值，以使`bar`包含`../`等字符序列，从而导致应用程序访问到意料之外的目录和文件。

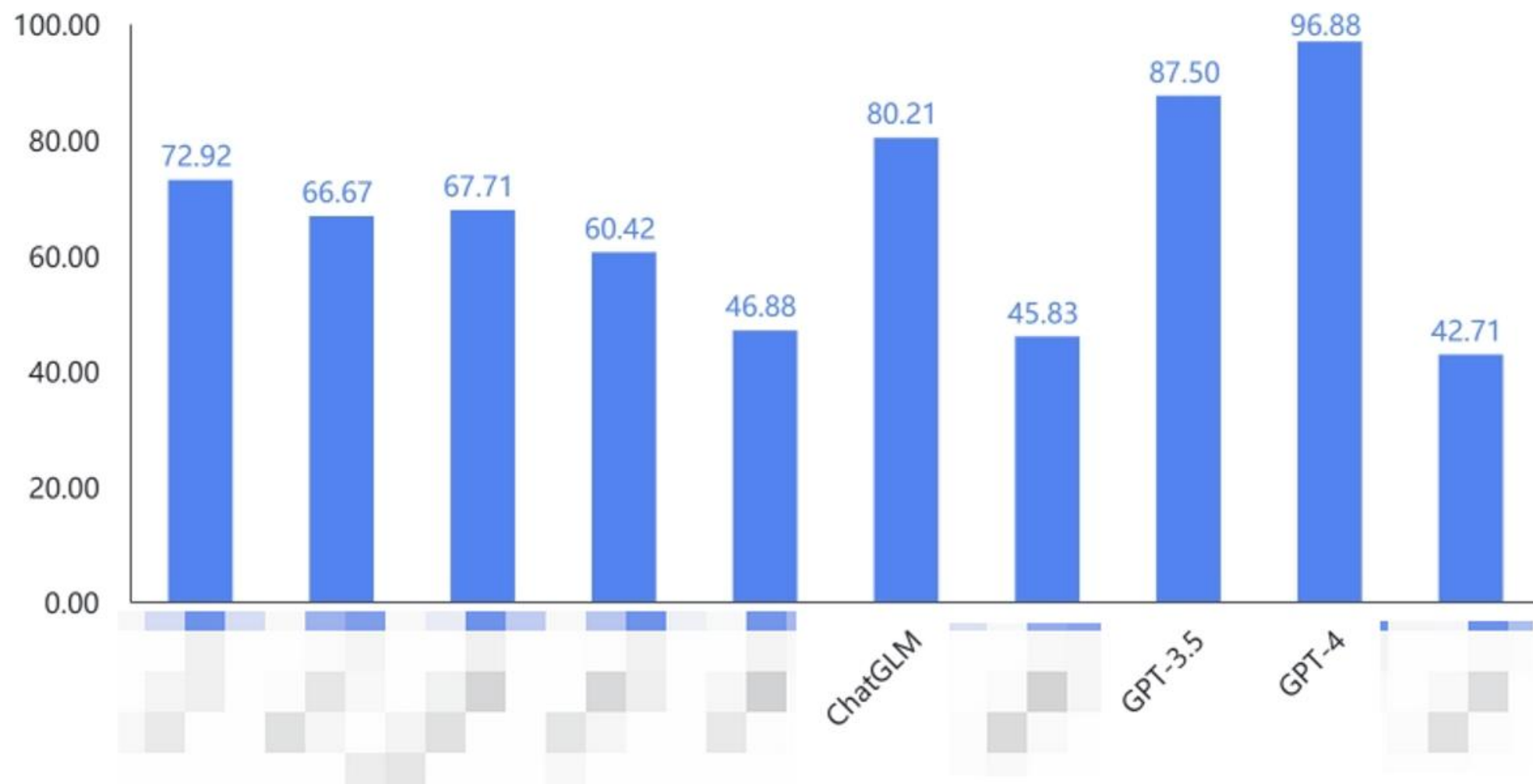
实验设计

- 选择最常见的两类变成语言Java与C
- 使用常用的Benchmark, OWASP JAVA Bench和NIST Juliet
- 使用大模型标准的公开版本API
- 丢弃上下文过长的测试例
- 使用相同的prompt
- 以发现BUG/漏洞为目标
- 人工判断大模型的输出结果是否正确

Benchmark-Java 大模型



Benchmark-C 大模型



一些结果分析

- 代码越简单，分析准确性越高
- 经常性遇到token窗口不足的情况
- 部分大模型存在特殊的字符过滤规则
- 缺乏推理/计算能力，复杂上下文的漏洞检出率很低
- 结果生成速度会是一个瓶颈

即使是简单的Benchmark(某些模型可能已经训练过)
检出率仍然不高，并且整个分析过程不可控

8. GPT-4 凸显的自回归架构的局限性

Limitations of autoregressive architecture highlighted by GPT-4

正如前面章节所展示的，GPT-4 在许多任务中表现出了令人印象深刻的能力，如推理、内容生成、问题解决等。然而，正如本节所示，**该模型也存在一些重大缺陷，其中一些似乎是自回归架构本质上的固有缺陷**。我们将通过一系列示例来说明这些缺陷，并讨论它们的影响。

GPT-4

Q: How many prime numbers are there between 150 and 250?

A: There are 13 prime numbers between 150 and 250.

然而，如果我们要求模型列出此范围内的质数并写出数量，它会给出正确的答案：

GPT-4

Q: First list the prime numbers between 150 and 250, and then output the size of this list.

A: The prime numbers are 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, and 241.

So the size of this list is 18.

8.2 算术/推理问题中的缺乏规划

8.3 文本生成中的缺乏规划

这些示例说明了下一个单词预测范例的一些局限性，这些局限性表现为模型缺乏规划、工作记忆、回溯和推理能力。**模型依赖于一种局部和简单粗暴地生成下一个单词的过程，没有任何关于任务或输出的全局或深层次理解**。因此，模型擅长产生流畅和连贯的文本，但在解决不能以顺序方式逐步解决的复杂或创造性问题方面存在局限。这指向了两种智力任务之间的区别：

目前来看，LLM 不适合用来做代码分析

LLM的优势

对话

解释

生成



解决代码安全治理的“最后一公里”

按规则按文件筛选

搜索规则

Cert C/C++ 标准(42)

类别02 - 声明与初始化(6)

类别03 - 表达式(7)

类别04 - 整型(7)

类别06 - 数组与容器(2)

不要形成或使用越界指针或数组下标(1)

0001 ARR30.c:10

确保可变量数组的大小参数在有效...

类别07 - 字符与字符串(2)

类别08 - 内存管理(6)

类别09 - 输入输出(2)

类别10 - 环境(1)

类别12 - 错误处理(8)

类别48 - 杂项(1)

文件 testcases/ARR30/ARR30.c 内

路径提示未审核

10 ← 循环回到循环的头部 →

11 ← 进入循环体 →

12 ← 循环回到循环的头部 →

13 ← 进入循环体 →

14 ← 循环回到循环的头部 →

15 ← 进入循环体 →

16 ← 循环回到循环的头部 →

17 ← 进入循环体 →

18 ← 循环回到循环的头部 →

19 ← 进入循环体 →

20 ← 循环回到循环的头部 →

21 ← 进入循环体 →

9 {

→ 10 ! j = t[i]; 22

22 ← 越界的内存访问

11 }

12 return j;

13 }

14

缺陷信息

路径 (1) 批注 修复建议 AI修复建议

类别06 - 数组与容器

[ARR30-C] 不要形成或使用越界指针或数组下标 高

CWE-119 CWE-125 CWE-129 ...

描述

不规范代码示例（形成越界指针）

在这个不规范的代码示例中，函数 f() 在把 index 用作 table 数组偏移量之前，验证了 index 小于 TABLESIZE。但是，该函数没有校验 index 为负数的情况。当 index 小于 0 时，函数返回语句中加法表达式的行为是未定义行为。在某些实现中，光这个数组加负数下标的行为（或对其解引用）可能触发硬件陷阱。而其他的实现中，数组加负数下标仍然可能产生一个可被解引用的指针，指向与原来数组 table 不同的对象。使用这个指针访问对象可能会导致信息泄露或数据篡改的错误。

解决方案

这个解决方案是检查并拒绝无效的 index：

```
enum { TABLESIZE = 100 };

static int table[TABLESIZE];

int *f(int index) {
    if (index < TABLESIZE) {
        return table + index;
    }
    return NULL;
}
```

```
enum { TABLESIZE = 100 };

static int table[TABLESIZE];

int *f(int index) {
    if (index >= 0 && index < TABLESIZE) {
        return table + index;
    }
    return NULL;
}
```

路径 (1) 批注 修复建议 AI修复建议

修复建议:

重新询问

这段代码存在安全问题，具体如下：1. 数组越界访问：在for循环中i的取值范围是0到10，但数组t的长度只有10，因此会出现数组越界访问，这可能导致程序崩溃或者出现不可预期的行为。2. 未初始化变量使用：在函数开头定义的变量j没有被初始化，然后在for循环中被赋值为t[i]。由于i的取值范围是0到10，因此当i等于10时，j的值未定义，可能导致程序崩溃或者出现不可预期的行为。针对这些问题，建议对代码进行改进：1. 将for循环的判断条件改为i<x，避免数组越界访问。2. 在for循环之前给变量j赋一个默认值，如j=0，避免未初始化变量使用的问题。

代码文件:

testcases/ARR30/ARR30.c

代码如下:

```
int testARR30()
{
    int x = 10;
    int j;
    int t[10] = {0};
    int i;
    for(i = 0; i<x; i++)
    {
        j = t[i];
    }
    return j;
}
```

红色部分是传统模板化修复建议

绿色部分是LLM生成的智能修复建议

黄色部分是LLM生成的修复代码

QCon
全球软件开发大会


InfoQ

1. 仍然会面对推理能力缺失/上下文窗口不足的问题



基于传统的程序分析能力，生成上下文信息，
优化prompt，缩小输入窗口，让场景收敛

```
public List<DictModel> queryAllTableDictItems(String table, String text, String code, String condition, String keyword) {  
    String filterSql = getFilterSql(table, text, code, condition, keyword);  
    List<DictModel> ls = baseMapper.queryAllTableDictItems(table, text, code, filterSql);  
    return ls;  
}
```



典型的SQL注入，涉及多个变量。从API入口到最终的SQL查询，有数层函数嵌套，上千行过程代码。

直接丢给大模型进行修复，会超过token上限，只给函数级代码会因为缺失上下文推理而生成似是而非的结果。

TA

请帮我修复以下代码中的注入漏洞：



```
public List<DictModel> queryAllTableDictItems(String table, String text, String code,
String condition, String keyword) {
    String filterSql = getFilterSql(table, text, code, condition, keyword);
    List<DictModel> ls = baseMapper.queryAllTableDictItems(table, text, code, filterSql);
    return ls;
}
```



SQL注入漏洞通常由于直接拼接用户输入的字符串到SQL查询中而造成的。为了避免SQL注入，你应该使用预编译的SQL语句或查询构造器，并避免直接拼接字符串。



在给出的代码中，`getFilterSql`函数可能会直接将用户输入的参数拼接到SQL查询中，这可能是一个潜在的风险。你应该修改这个函数以使其安全，或者使用其他方式来安全地生成SQL。

这里是一个修复的建议：

1. 使用预编译的SQL语句：

这里假设`baseMapper.queryAllTableDictItems`是一个使用MyBatis或类似的ORM框架的方法。你应该修改其SQL映射文件以使其使用预编译的语句。

xml



Cop



Regenerate

TA

In the following code, line 3 has a SQL injection vulnerability, the first and second parameter can be control by the user input, please help me fix it:

```
public List<DictModel> queryAllTableDictItems(String table, String text, String code, String condition, String keyword) {  
    String filterSql = getFilterSql(table, text, code, condition, keyword);  
    List<DictModel> ls = baseMapper.queryAllTableDictItems(table, text, code, filterSql);  
    return ls;  
}
```



You're right. If the `'table'`, `'text'`, or `'code'` parameters are directly coming from user input, there's a potential SQL injection vulnerability. Here's how you can fix the code to avoid SQL injection:

1. Use parameterized queries or prepared statements whenever possible. This is the most recommended approach as it handles the SQL escape for you and doesn't let any malicious input execute as SQL.
2. If the above isn't possible due to the nature of your dynamic table or column names, then you must use a strict whitelist approach. This means that you only allow known good table names or column names to be used.

Here's a revised version of your code:

java

Copy code

```
public List<DictModel> queryAllTableDictItems(String table, String text, String code, String condition, String keyword) {  
    // Step 1: Validate against a whitelist.  
    if (!isValidTableName(table) || !isValidColumnName(text) || !isValidColumnName(code)) {  
        throw new IllegalArgumentException("Invalid table or column name");  
    }  
    String filterSql = getFilterSql(table, text, code, condition, keyword);  
    List<DictModel> ls = baseMapper.queryAllTableDictItems(table, text, code, filterSql);  
    return ls;  
}
```

Regenerate

Send a message

- 基于已有的静态代码分析能力，定位问题所在代码行和漏洞类型
- 基于程序分析的结果，还原控制流/数据流信息
- 压缩上下文
- 填入模版，生成prompt
- 指导大模型做出正确的解释，生成我们需要的修复代码。

2. 提升编程类大模型安全相关任务的推理能力



专用的数据集来重新fine-tuning甚至重新训练模型

模型

商用许可

推理成本低

本地化部署

底子好

数据集

基于分析工具生成

历史数据积累

爬虫+启发式标记

人工标记

GPT-4清洗

代码安全治理的“最后一公里”

Coding



研发

触发扫描



Jenkins



判定结果



研发/安全

修复代码



研发/安全

复盘培训



研发/安全

智能提示

精准分析

智能解释

自动修复

团队赋能

THANKS

—
软件正在重新定义世界

Software Is Redefining The World

