北京理工大学
信息系统及安全对抗实验中心
信息安全与对抗技术研究所
Beijing Forest Studio

智能体辅助的协议模糊测试

www.isclab.org.cn

硕士研究生 徐菊彬

2026年09月20日

问题回溯

- 问题回溯
  - 讲授时长分配不合理，对基础概念讲授过多
  - 部分页面ppt排版美观度不足，仅文字
- 相关内容
  - 2024.09.03 张浩然《大模型赋能的模糊测试用例生成技术》
  - 2024.12.23 徐菊彬《网络未知协议逆向技术》
  - 2025.05.18 徐菊彬《大模型指导的协议模糊测试》
  - 2025.06.30 张浩然《软件灰盒定向模糊测试技术》
  - 2026.02.08 徐菊彬《协议模糊测试方法》
  - 2026.03.16 杨语航《基于智能体的自动化漏洞重现方法》

内容提要

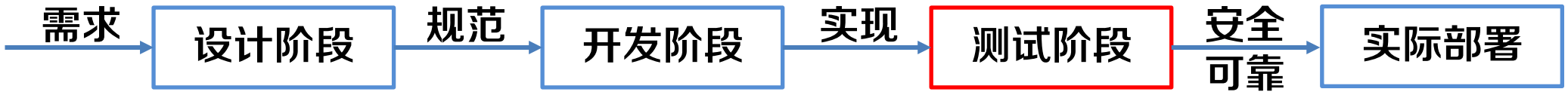
- 预期收获
- 题目内涵解析
- 研究背景与意义
- 研究历史与现状
- 知识基础
- 算法原理
  - ProtocolGuard
  - BSFuzzer
- 特点总结与工作展望
- 参考文献

预期收获

- 预期收获
  - 1. 了解智能体在协议模糊测试方法中的应用
  - 2. 理解两篇模糊测试方法的基本原理与区别
  - 3. 思考智能体技术对协议模糊测试的贡献

内涵解析与研究目标

- 内涵解析
  - 网络协议：计算机网络中进行数据交换的规则，保证节点之间的相互通信



  - 模糊测试：生成大量随机突变测试用例，旨在触发软件程序中异常运行时的行为
  - 协议模糊测试：通信复杂性高、测试环境相对受限
  - 智能体：以LLM为核心，围绕给定目标进行任务理解、规划决策、工具调用和环境交互，并根据执行结果持续调整后续行为的智能系统
    - 感知、决策、执行、验证、循环反馈
- 研究目标
  - 提高协议模糊测试的漏洞检出能力
    - LLM提供语义理解与生成能力
    - 智能体组织工具执行和反馈修正

研究背景与意义

- 研究背景
  - 网络协议是互联网通信的基础，协议实现的复杂性使得开发过程中易引入漏洞，导致严重的安全问题
    - Heartbleed漏洞存在于OpenSSL的TLS协议实现中，在实现TLS的心跳扩展时没有对输入进行适当验证（缺少边界检查），导致缓冲区过读，影响了全球超过17%的服务器
  - 传统模糊测试在协议场景中具有局限性，需针对协议规范设置结构化测试用例，实现状态空间的有效探索
- 研究意义
  - 研究高效的协议模糊测试技术，在软件发布前或部署早期主动发现漏洞，提升网络基础设施安全性与健壮性

研究历史与现状

Pham等人提出AFLNET，第一个用于协议实现的灰盒模糊器，将消息序列视为覆盖定向灰盒模糊测试的种子，将每条消息与相应的协议状态相关联，并最大化状态空间和代码的覆盖范围，是首个使用代码和状态覆盖引导的协议模糊器

Meng等人提出大模型指导的模糊测试方法CHATAFL，将大语言模型与经典协议模糊测试器AFLNet相结合，利用LLMs学习协议规范文档RFC，指导生成合规且多样化的输入数据，同时设置覆盖高原检测机制，提高了覆盖率和漏洞发现效率

Yang等人提出了一个面向低功耗蓝牙协议（Bluetooth Low Energy，BLE）的逻辑缺陷检测的黑盒模糊测试框架。利用LLM代理解析蓝牙规范，提取状态机、字段语义与跨报文依赖，实施上下文告知的字段和状态变异，分析设备响应，识别非崩溃逻辑缺陷

2020 2023 2024 2025 2026 2026

Qin等人提出基于程序变量的状态表示方案和有效的交互同步机制来提高模糊测试效率NSFuzz，通过手动注释消息处理的结束来减少延迟，使用静态分析和注释应用程序编程接口（API）来识别服务内的同步点和状态变量，实现快速 I/O 同步和准确的服务状态跟踪

Xia等人提出HSPFuzzer，一种高速协议模糊器，它利用连接重用来减少 SUT 重新启动和连接重新建立。采用前缀消息识别算法来确定连接内所需的基本数据包，并采用了覆盖监控机制来检测异常执行状态

Song等人提出ProtocolGuard，一种结合LLM引导静态分析与动态验证的协议规范不一致缺陷检测框架。通过规则提取与程序切片分析规范和实现的偏差，利用智能体生成断言及测试脚本，将静默违规转化为断言失败，再通过定向协议模糊测试获取触发证据与复现用例

知识基础 协议模糊测试

- 通用模糊测试
  - 生成大量随机输入数据（测试用例），提供给目标程序运行
  - 检测程序在异常输入下的行为
  - 高效检测目标程序中任意位置存在的漏洞
- 定向模糊测试（Directed Fuzzing）
  - 在模糊测试的基础上，将更多测试资源放在目标特定位置
- 协议模糊测试的对象是协议实现源代码，基本等同于软件模糊测试
  - 提高漏洞检出能力，覆盖更多协议实现源码，缺陷位置可达
  - 测试用例符合字段语义、状态转换等协议规范

知识基础 黑盒、灰盒、白盒模糊测试

- 黑盒模糊测试
  - 仅能获取程序的输入输出及外部行为
    - 协议响应、响应时间、连接中断、程序崩溃现象
  - 不依赖程序内部实现，通过发送测试输入，观察响应等外部行为来发现缺陷
- 灰盒模糊测试
  - 除外部反馈外，还能获取有限的内部运行信息　　执行情况监控
    - 代码覆盖率、分支执行情况等
  - 利用代码覆盖率等有限的运行时信息，引导输入变异，探索尚未覆盖的执行路径
- 白盒模糊测试
  - 能获取并分析源代码中的控制流、数据依赖和路径约束等程序内部逻辑
  - 通常结合符合执行与约束求解，生成能够触发特定路径的测试输入

BSFuzzer: Context-Aware Semantic Fuzzing for BLE Logic Flaw Detection

算法原理 TIPO

T	目标	提高对低功耗蓝牙（Bluetooth Low Energy， BLE）协议的逻辑缺陷检出能力，黑盒协议模糊测试
I	输入	协议规范文档
P	处理	1. 语义解析：提取状态机、字段语义及跨报文依赖 2. 测试序列生成：字段级、状态级变异 3. 错误验证：依据协议规范分析响应偏差
O	输出	漏洞报告

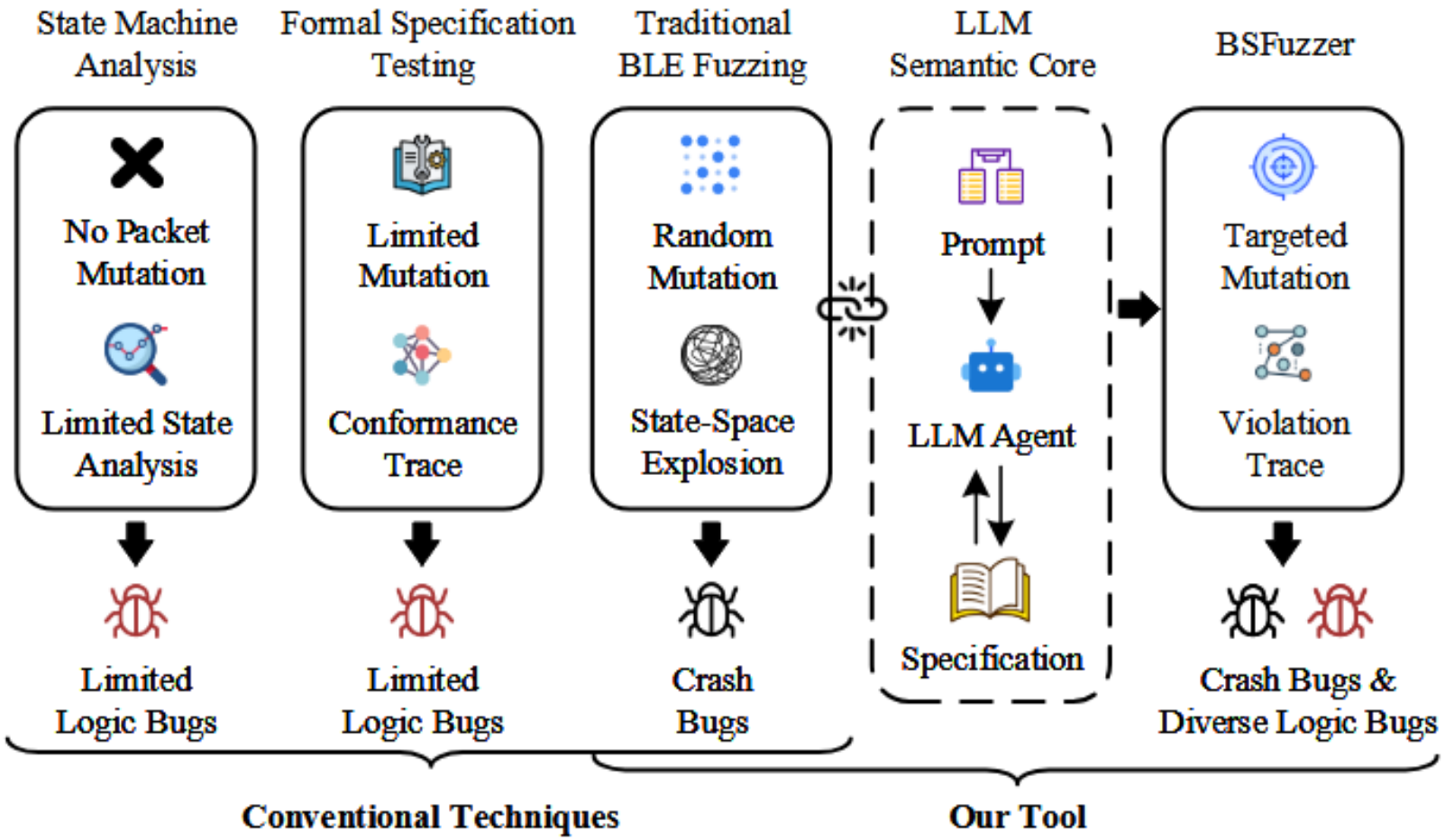
P	问题	无约束变异容易破坏会话前提，难以触及深层逻辑； 传统崩溃判定容易遗漏非崩溃缺陷，人工设计测试与分析响应成本高
C	条件	协议规范文档可获取
D	难点	准确提取跨报文与状态依赖； 区分正常拒绝、链路异常和真实逻辑违规
L	水平	NDSS2026 CCF A

算法原理 科学问题 应用问题

- 科学问题
  - BLE消息的处理有效性与响应正确性共同依赖前置状态、跨报文字段关系
  - 无约束变异易破坏执行前提，使测试无法触及目标逻辑
  - 单个响应或崩溃信号不足以判断实现是否违反规范
- 应用问题
  - 现有方法对字段语义处理错误、非法状态转换等逻辑缺陷的检出能力不足
    - 逻辑缺陷在有效协议执行期间出现，不会导致崩溃或可观测故障
  - 测试结果的判定与确认依赖人工

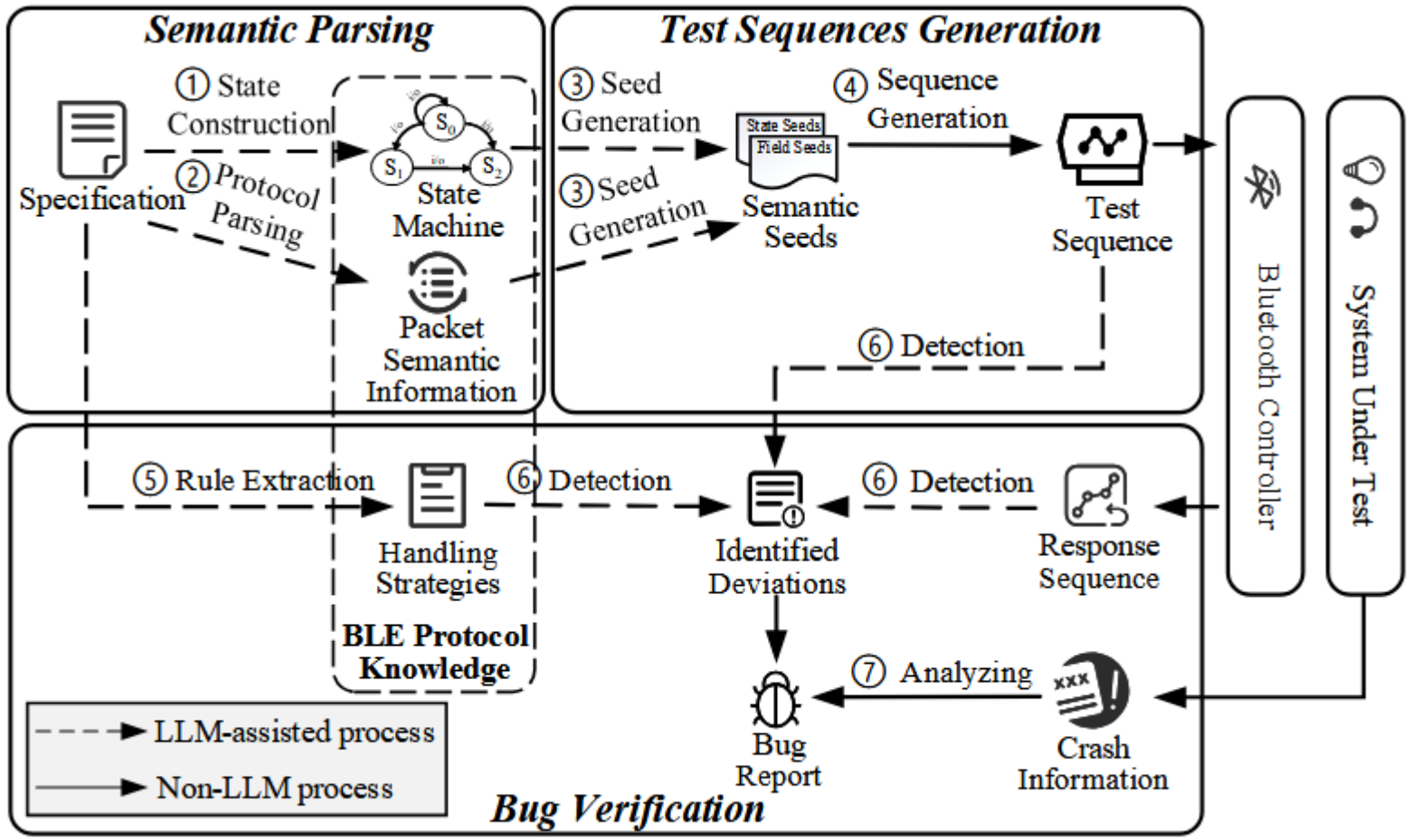
算法原理 创新说明

- 将智能体集成到传统模糊测试方法
  - 从规范文档中理解协议语义与状态转换
  - 字段、状态变异
    - 有意违反目标约束
    - 预测违规后的行为
  - 模糊测试收集反馈
    - 比较预期行为与实际响应
  - 分析bug



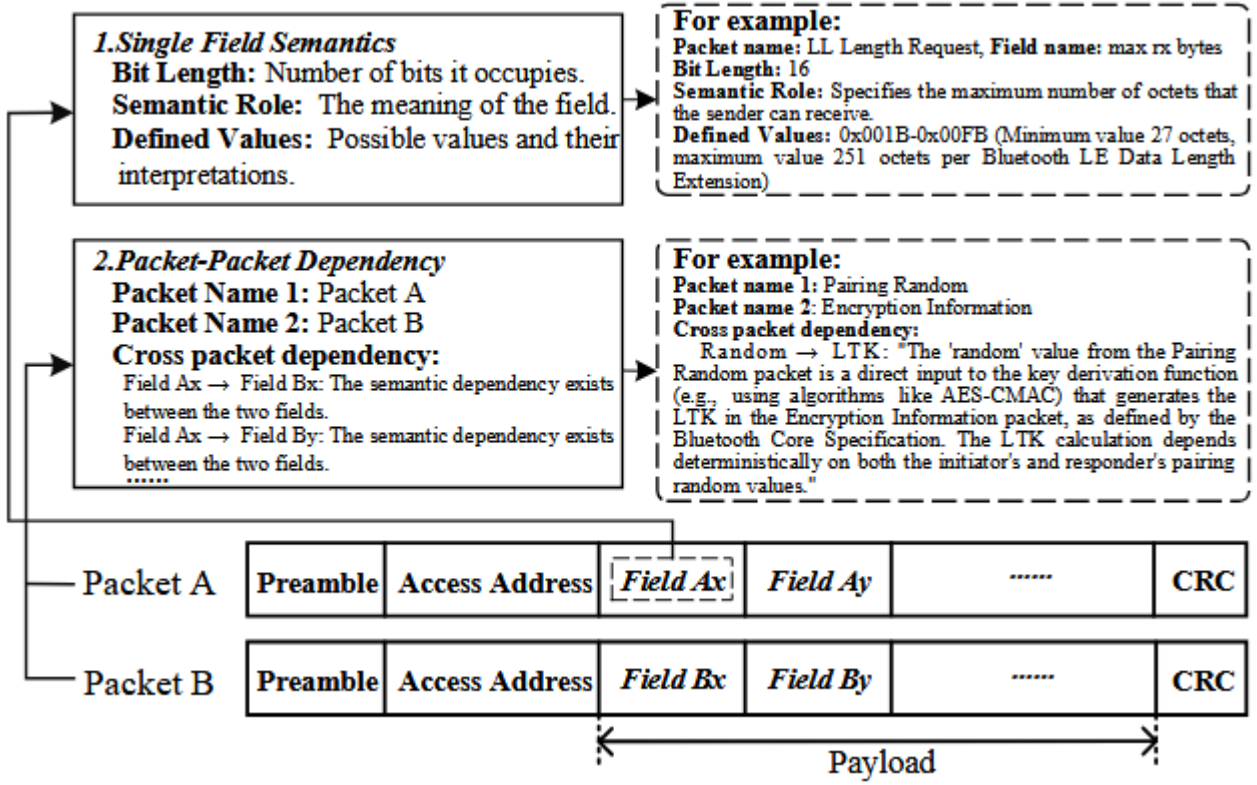
算法原理 BSFuzzer

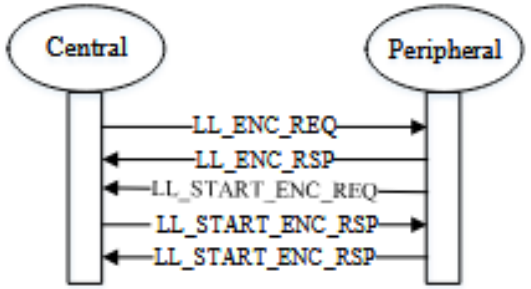
- 核心算法原理
  - 语义解析
    - 协议状态机信息
    - 数据包语义信息
  - 测试序列生成
    - 语义种子生成
    - 测试序列合成
  - Bug验证
    - 提取协议规则
    - 检查预期行为与实际设备响应是否存在偏差



算法原理 BSFuzzer 语义解析

- 协议状态机构造
  - 使用智能体解析协议规范中定义的消息序列图（Message Sequence Charts, MSCs）
  - 提取底层协议逻辑并构造相应的协议状态机





消息序列图

Current State	Input/Output	Next State
S0	LL_ENC_REQ / LL_ENC_RSP, LL_START_ENC_REQ	S1
S1	LL_START_ENC_RSP / LL_START_ENC_RSP	S2

状态机构建

- 数据包语义解析
  - 从规范中提取单字段语义和数据包依赖关系2种语义信息

算法原理 BSFuzzer 语义解析

- 数据包语义信息解析
  - 单字段语义
    - 结构化提示智能体为每个字段提取3个关键语义属性
      - 位长度，结构正确性
      - 语义角色，功能目的
      - 定义值，有效范围/特殊常量
  - 数据包依赖关系
    - 遵守协议定义的语义
    - 仅检测直接依赖关系
      - 某个字段直接决定另一个字段

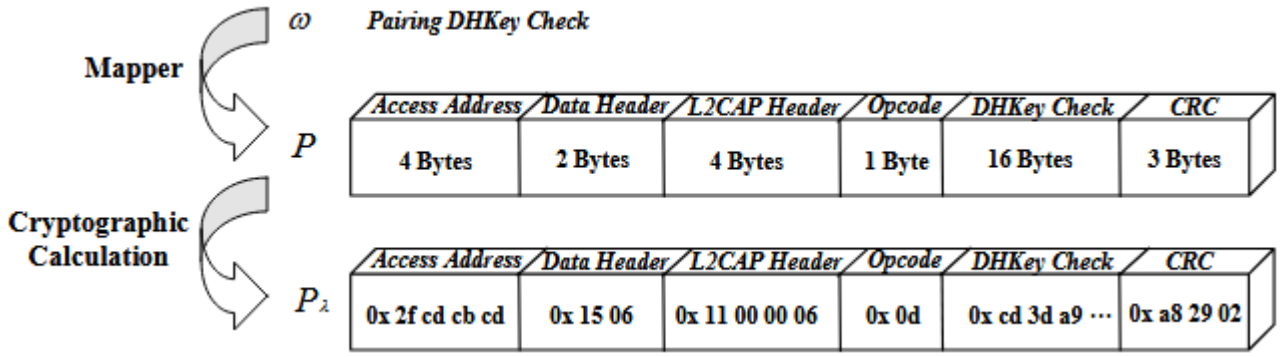
Prompt Template for Packet-Packet Dependency

As a Bluetooth protocol expert, you need to analyze the direct dependencies between the two data packet parameters of **[Packet_1]** and **[Packet_2]**.
Please follow these steps for professional analysis:
Data packet definition:
ALL **[Packet_1]** field: **[Packet_1_fields]**
ALL **[Packet_2]** field: **[Packet_2_fields]**
Analysis requirements:
1. Strictly analyze based on the provided **[Packet_1]** fields and **[Packet_2]** fields without omitting or fabricating any fields.
2. Direct dependency is defined as: The parameter value in one data packet directly affects or determines the setting of the parameter value in another data packet.
3. Consider the constraints between the parameters defined by the protocol specification.
4. Exclude indirect effects or associations generated by intermediate parameters.
Complete it in the <think> tag:
1. List the protocol definition of each parameter
2. Analyze the direct causal relationship between the parameters
3. Record the excluded indirect associations
Output requirements:
1. Use strict JSON format
2. Contain Boolean cross_packet_dependency fields
3. Describe each parameter pair relationship with key-value pairs in the explanation field
4. Keeping the same field structure as the example

[Example Output]

算法原理 BSFuzzer 测试序列生成

- 利用智能体生成测试序列，触发逻辑缺陷
- 种子生成
  - 字段突变　规范性
    - 使用结构化提示让智能体生成语义引导突变
  - 状态突变　可达性
    - 根据规范导出的状态机和数据包依赖性
    - 利用智能体围绕目标数据包生成测试路径
- 序列生成



Generation Rules:
1.Parse Dependent packets to get the state transition requirements related to **[Packet]**.
2.Comprehensive coverage of **[Packet]**-related protocol violations to ensure comprehensive testing.
3.Based on the state transition requirements, generate the packet transmission path, at least one of the following state dependencies must be violated:
-Must include: **[Packet]**
-Prestate Not Completed
-Repeated Operations Not Following Protocol Specifications
-Unexpected State Change
-Combinations containing the above variants
-Each test case corresponds to a specific failure scenario
-Technical basis description (cite the protocol chapter or specification clause)

Generation Steps:

1. List violations of specific parts and descriptions related to **[Packet]** in `<Analysis>` tags.
2. Generate the packet transmission path.
3. Only focus on the **[Packet]** related state transition mutation.
3. Confirm in the `<Validation>` tag whether each mutation directly violates protocol clauses.
4. Describe the specific mutation steps in `<Mutation Steps>`, using only: - Send [data packet name] - Skip [data packet name].
5. Generate xx test cases, no repeat.

算法原理 BSFuzzer Bug验证

- 依据规范判断设备响应是否违规
- 字段与状态处理规则提取
  - 利用智能体提取字段&状态规则
  - 预测字段级、状态级违规后的协议响应
- 预期行为与实际响应的一致性检测
  - 字段验证
    - 确定变异字段值是否符合规范
    - 检查设备响应是否符合定义的错误处理逻辑
  - 状态验证
    - 智能体逐包验证设备是否处于有效协议状态
    - 分析实际响应和预期行为间的差异
- Bug分析

Prompt Template for State Validation

You are a Bluetooth Core Specification expert. Given a sequence of BLE packets: **[Send_Sequence]** and **[Device_Response_Sequence]**, perform the following checks for **[Send_Packet]** in the path:

1. Pre-State Verification

For **[Send_Packet]**, determine whether the device was in a valid protocol state to legally receive and process this packet, based on the **[Precondition]**.

If the send packet is expected in the current state, it should be treated as a state-valid packet.

If the send packet is unexpected in the current state, it should be treated as a state-invalid packet.

2. Device Response Compliance

For response packet:

If the pre-state was valid, verify whether the device replied with the expected **[Valid_Response]**.

If the pre-state was invalid, verify whether the device replied with the expected **[Invalid_Response]**.

[Example Output]

实验设计　数据资源

- 数据资源
  - 9个BLE供应商
  - 10款智能手机

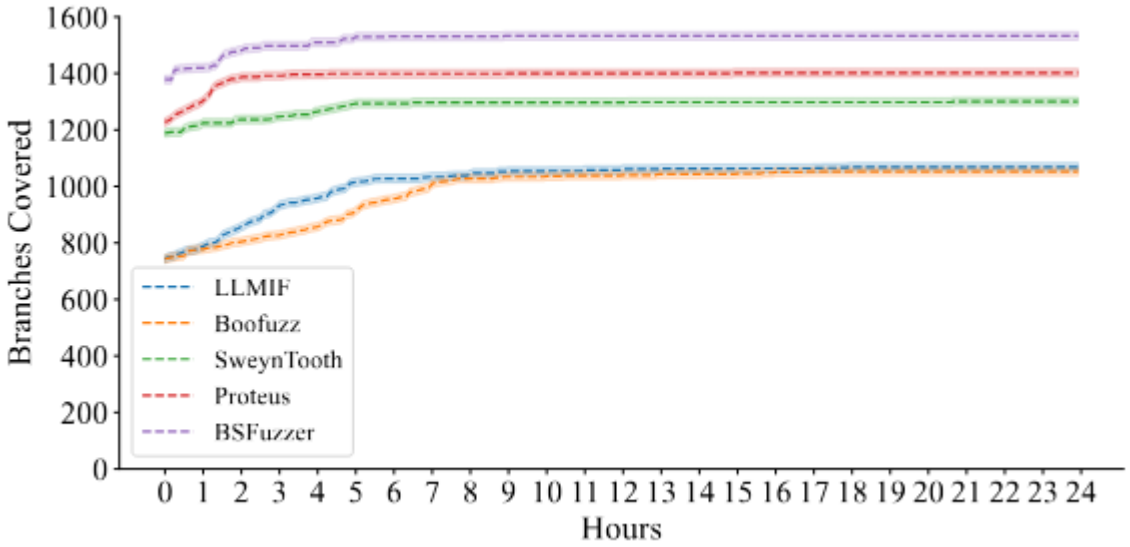
Category	ID	Device	Vendor/Manufacturer	BLE Ver.
BLE SoC	D1	CY8CKIT-042-BLE	Cypress	5.1
	D2	WBZ451	Microchip	5.2
	D3	ESP32-WROOM-32E	Espressif	4.2
	D4	B91	Telink	5.0
	D5	RTL8762EKF-EVB	Realtek	5.0
	D6	CH592	WCH	5.3
	D7	LP-CC2652RB	Texas Instruments	5.2
	D8	nRF52840-DK	Nordic Semiconductor	5.0
	D9	NUCLEO-WB55RG	STMicroelectronics	5.2
Smartphone	D10	P40	Huawei	5.1
	D11	Nova5	Huawei	5.0
	D12	Mate50	Huawei	5.2
	D13	Honor90	Honor	5.2
	D14	OPPO A72	OPPO	5.0
	D15	Note 10 Pro	Redmi	5.0
	D16	Xiaomi 14	Xiaomi	5.4
	D17	Galaxy C55	Samsung	5.2
	D18	Google Pixel 2	Google	5.0
	D19	Google Pixel 6 Pro	Google	5.2

- 对比方法
  - Boofuzz（2024）
    - 面向IoT协议的通用黑盒模糊方法
  - LLMIF（2024）
    - 集成LLM面向Zigbee协议的黑盒模糊测试方法
  - SweynTooth（2020）
    - BLE协议专用模糊器
  - Proteus（2024）
    - 基于状态机突变的无线通信协议测试框架
- 评价指标
  - 漏洞检出能力、代码覆盖率

实验结果　对比实验

- 漏洞检出能力优于对比方法
  - Boofuzz&LLMIF
    - 缺乏对 BLE 特定状态转换的支持，无法探索深层协议逻辑并发现与状态相关的错误
  - SweynTooth
    - 缺乏适当的测试预言机，无法识别与状态相关的问题
  - Proteus
    - 仅限于BLE核心规范中的预定义属性，仅识别明确指定的违规行为，同时遗漏需要理解协议状态、字段语义和行为逻辑的深层逻辑漏洞

- 代码覆盖率较Proteus提高9.34%



M：Memory corruption/ S：State bugs/ I：inconsistency bugs

Device	Boofuzz			LLMIF			SweynTooth			Proteus			BSFuzzer		
	M	S	I	M	S	I	M	S	I	M	S	I	M	S	I
CY8CKIT-042-BLE	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1
WBZ451	1	0	0	1	0	0	1	0	0	0	0	0	1	1	1
ESP32-WROOM-32E	1	0	0	1	0	0	1	0	1	0	1	1	1	2	2
RTL8762EKF-EVB	1	0	0	1	0	0	1	0	0	0	1	0	1	3	0
SUM	3	0	0	3	0	0	4	0	0	0	2	1	4	7	4

实现结果 漏洞发现

- 现实世界中Bug发现
  - M：内存损坏，S：状态错误，I：不一致错误

ID	Bug Description	Pair.	Issue	CVSS	Security Impact Summary	Spec. Ref.
M1	Buffer overflow via oversized `len` in `LL_PAUSE_ENC_REQ`	Leg.	DoS	6.5	Causes crash due to unchecked field size	–
M2	Buffer overflow via RFU=7 in `LL_TERMINATE_IND`	–	DoS	4.3	Improper handling of reserved value leads to memory fault	–
M3	Invalid `ChMap`=0x0 in `CONNECT_REQ`	–	DoS	6.5	Triggers crash during setup, preventing new connections	V6B§4.5.8
M4	Invalid `AccessAddr`=0x0 in `CONNECT_REQ`	–	DoS	6.5	Triggers crash during setup, preventing new connections	V6B§2.1.2
M5	Invalid `timeout`=0xFFFF in `CONNECT_REQ`	–	DoS	6.5	Triggers crash during setup, preventing new connections	V6B§4.5.2
M6	Invalid `win_offset`=0xFFFF in `CONNECT_REQ`	–	DoS	6.5	Triggers crash during setup, preventing new connections	V6B§2.3.3
S1	Accepts `LL_PAUSE_ENC_REQ` before encryption is enabled	–	DoS	7.5	Allows early transition to encryption pause state	V6B§4.7
S2	SK reuse allows reconnection without re-pairing	SC	SecByp	8.8	Enables attacker to rejoin session using old SK	V3H§2.3.5
S3	Accepts zero LTK after `PAIRING_FAILED`	Leg.	SecByp	8.1	Neglects key validation; encryption proceeds with all-zero key	V3H§2.4.4
S4	Misaligned state by early `LL_ENC_REQ` before pairing	–	DoS	7.5	Improper sequence handling leads to desync	V3H§2.4
S5	Accepts `LL_START_ENC_RSP` before pairing	–	DoS	8.1	Fails to verify preconditions before encryption start	V6B§4.7
S6	Multiple `LL_LENGTH_REQ` causes crash	–	DoS	6.5	Flooding `LL_LENGTH_REQ` causes stack overflow	V6B§4.5.10
S7	Accepts `LL_START_ENC_REQ` before pairing	–	DoS	7.5	Bypasses authentication phase during pairing	V6B§4.7
S8	Accepts `PAIRING_RANDOM` before public key exchange	SC	DoS	7.5	Breaks pairing sequence	V3H§2.3.5.7
I1	Accepts `LL_LENGTH_REQ` with `max_tx_bytes` < 27	–	DoS	6.5	Violates minimal payload constraint	V6B§4.5.10
I2	Falls back to Legacy despite SC request	SC	Downg.	7.6	Allows downgrade to weaker authentication mode	V3H§2.2
I3	Incorrect response to malformed `LL_PAUSE_ENC_REQ` with invalid RFU	–	–	–	–	V6B§4.7
I4	Incorrect response to malformed `LL_LENGTH_REQ` with invalid params	–	–	–	–	V3H§2.3.5.10
I5	Disconnect triggered by premature `LL_PAUSE_ENC_REQ`	–	DoS	4.5	State inconsistency during encryption setup	V6B§4.7

Note: Leg.=Legacy Pairing; SC=Secure Connections; DoS=Denial of Service; SecByp=Security Bypass; Downg.=Security Downgrade. "–" means no observable security impact. "V6B§4.7" denotes Volume 6, Part B, Section 4.7 of the specification.

算法总结

- 算法贡献
  - 提出用于BLE协议栈的上下文感知语义模糊框架，检出逻辑和状态的不一致错误
    - 利用智能体提取协议知识，预测执行响应
    - 进行模糊测试，比较预期行为与实际反馈是否存在不一致

- 算法不足
  - 由于BLE协议本身具有硬件限制，评估仅限于*Just Works*身份验证方法
    - 其他身份验证机制方法导致的潜在漏洞未被发现
  - 处理复杂协议的状态转换时，LLM结果发生误判或忽视潜在问题
    - 需要人工分析验证

ProtocolGuard: Detecting Protocol Non-compliance Bugs via LLM-guided Static Analysis and Dynamic Verification

算法原理 TIPO

T	目标	检测协议实现（代码）与协议规范（RFC文档）间的不合规错误
I	输入	1. 待测系统：11种开源网络协议实现源码（Sol、TinyMQTT、Mosquitto、libcoap、FreeCoAP、pure-ftpd、uFTP、TLSE、wolSSL、Dnsmasq、NDHS） 2. 相关协议规范RFC文档
P	处理	1. 协议规则提取 2. LLM引导的程序切片 3. 基于LLM的不一致性检测（静态分析） 4. 使用定向模糊测试进行动态验证
O	输出	PoC（Proof-of-Concept）、漏洞报告

P	问题	1. 不合规错误发生时无明显崩溃信号，传统模糊测方法难以检测 2. 现有方法常人工验证结果并分析根本原因，自动化程度不高
C	条件	协议实现源码可获取，具有公开协议规范文档
D	难点	如何在没有明确错误信号的情况下有效验证不合规错误
L	水平	NDSS2026 CCF A

算法原理 科学问题 应用问题

- 科学问题
  - 自然语言表述的协议规范与协议实现的代码存在语义偏差
  - 语义偏差不会导致程序运行过程中出现中断等明显的崩溃现象
- 应用问题
  - 现有模糊测试方法对于协议实现中与协议规范不一致的缺陷，漏检率高
    - 此类缺陷多为不触发程序崩溃的逻辑缺陷，不产生显示异常
  - 且缺陷验证、根因分析等任务常依赖人工，自动化实现不充分

如何准确关联规范语义与代码实现逻辑，
将隐式的违反协议规范的代码实现转化为可检测的执行证据

算法原理 创新说明

- 结合LLM引导的静态分析与基于模糊测试的动态验证，检测不合规错误

- 关联协议规范与协议实现
  - 协议规范提取
  - LLM引导的程序切片
  - 基于LLM的不一致性检测
- 利用模糊测试进行缺陷验证与PoC生成
  - 断言生成&检测
    - 使用智能体迭代生成断言并检测
  - 测试用例生成
    - 使用LLM生成语法有效且违反协议规则的测试用例
  - 定向模糊测试

```
1  struct client {
2    ...
3    char client_id[MQTT_CLIENT_ID_LEN];
4  }
5  static int connect_handler(struct io_event *e) {
6    struct mqtt_connect *c = &e->data.connect;
7    struct client      *cc = e->client;
8
9    // BUG: Client ID truncation without validation
10   snprintf(cc->client_id, MQTT_CLIENT_ID_LEN, "%s
       ", c->payload.client_id);
11   ...
12 }
```

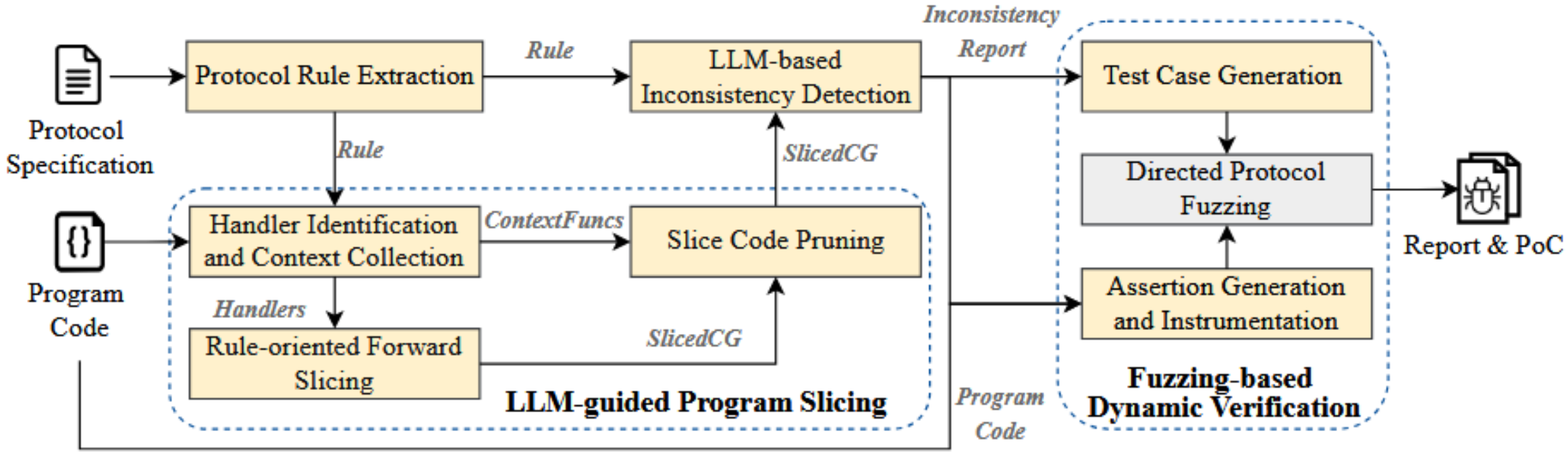
snprintf将客户端ID复制到固定大小缓冲区中

缺乏长度验证，服务器会截断超出限制的字符

导致多个客户端共享相同的截断标识符

算法原理 ProtocolGuard

- **ProtocolGuard**
  - **LLM辅助的静态分析**
    - **协议规则提取、程序分片、不一致性检测**
  - **动态验证（定向模糊测试）**



算法原理 ProtocolGuard 协议规则提取

- 从规范文档RFC中提取协议规则
- 关键字匹配进行候选规则识别
  - 协议关键字（消息、字段名称）
    - 将wireshaek解析器代码作为上下文
  - 模态关键字（必须、应该）
  - 数值比较关键字（大于、小于）
  - LLM扩充关键字

```
{
  "rule": "The Server MUST allow ClientIds which
    are between 1 and 23 UTF-8 encoded bytes in
    length, and MAY allow ClientIds that contain
    more than 23 encoded bytes",
  "req_type": "CONNECT",
  "req_fields": ["Payload", "ClientId"],
  "res_type": "CONNACK",
  "res_fields": []
}
```

- 候选规则内容扩充

If this extension is present in the ClientHello, servers MUST NOT use…

  - 单个候选规则子句语义不完整
  - LLM分析每个候选规则的周围部分，合并逻辑上连接的子句
- 结构化规则表示
  - Json表示
    - 原始文本规则、请求消息类型、内部字段、响应消息类型和字段

算法原理 ProtocolGuard 程序切片

- 提取与每个规则相关的代码切片
- 定位消息处理逻辑
  - 构建消息相关调用图
  - LLM定位规则对应的消息处理函数
  - 收集上游连接、资源管理上下文
- 追踪相关数据并补全代码
  - LLM识别规则字段对应的变量
  - LLVM追踪变量的辅助、使用与跨函数传递
  - AST补全结构控制，LLM补充语义信息
- 语义剪枝与合并
  - 裁剪无关逻辑，保留必要调用关系
  - 合并上游上下文，生成最终代码切片

Algorithm 1 Workflow of LLM-guided Program Slicing

Input: *M* (LLVM Module), *Rule*, *LLM*
Output: *CodeSlice*

// Phase 1: Handler Identification and Context Collection
MessageCG ← ExtractMessageProcessingSubgraph(*M*)
Handlers ← IdentifyMessageHandlers(*MessageCG*, *Rule*, *LLM*)
ContextFuncs ← IdentifyContextFunctions(*MessageCG*, *Handlers*, *Rule*, *LLM*)
// Phase 2: Rule-oriented Forward Slicing
SlicingCriteria ← IdentifySliceCriterion(*Handlers*, *MessageCG*, *Rule*, *LLM*)
if *SlicingCriteria* $\neq \emptyset$ **then**
    InitialSlice ← ForwardSlice(*MessageCG*, *SlicingCriteria*)
    SlicedCG ← CompleteSliceCode(*Handlers*, *InitialSlice*, *LLM*)
else
    SlicedCG ← *MessageCG*
end if
// Phase 3: Slice Code Pruning
PrunedCG ← PruneIrrelevantCode(*Handlers*, *SlicedCG*, *Rule*, *LLM*)
CodeSlice ← GenerateCodeSlice(*M*, *PrunedCG*, *ContextFuncs*)

算法原理 ProtocolGuard 不一致性检测

- LLM不一致性检测
  - 解析规则要求
  - 考察代码片的控制流程和数据处理逻辑
  - 确定代码实现是否满足规则的要求
  - 结构化输出不一致报告

```
{
 "result": "violation found!",
 "reason": "..., the critical violation occurs in
     `connect_handler` where externally provided
    client IDs are truncated.",
 "violations": [
  {"function_name": "connect_handler", "filename
    ": "/path/handlers.c", "code_lines": [394]}
 ]
}
```

#Instruction
Your task is to analyze the provided code slice (#SliceCode) to determine whether it violates the constraints or behaviors specified in the rule description (#Rule) for **{protocol_name} {protocol_version}**, i.e., whether the developer's implementation adheres to the prescribed standards.

Please follow these steps for the analysis:
1. Carefully review the specific requirements and constraints described in #Rule.
2. Examine the key logic and behavior in the code slice (#SliceCode). Note: the numbers in the first column of each line in #SliceCode are line numbers, used to identify the code's position in the file.
3. Determine whether the code slice violates any requirements in the rule description. Strictly adhere to #Rule without referencing any external content.
4. If a violation is found, provide a detailed explanation, including the reason for the violation, the relevant code lines, and their associated filenames and function names.
5. If no violation is found, explain why the code correctly complies with the rule.
6. Follow the response template below:
If a violation is found, return the following JSON response:

```
{
 "result": "violation found!",
 "reason": "detailed reasons for violations",
 "violations": [{
    "function_name": "function name",
    "filename": "/path/filename",
    "code_lines": [line_number, ...]},...]
}
```

If no violation is found, return the following JSON response:

```
{
 "result": "no violation found!",
 "reason": "why the code correctly follows to the rule"
}
```

#Rule
{rule_desc}
#SliceCode
{code_snippet}

算法原理 ProtocolGuard 动态验证

- 使用定向模糊测试技术验证不一致性结果并生成相应的PoC
- 智能体生成并修正断言语句
  - 分析：提取检查条件，定位插桩位置
  - 执行：检索源码，生成断言与辅助函数
  - 反馈：调用编译工具，读取报错并修改代码
- 测试用例生成
  - 根据协议规则与不一致报告，生成候选反例描述
  - 利用智能体生成python脚本，构造PCAP报文
  - 调用流量分析工具来验证PCAP报文是否与预期消息匹配
- 定向模糊测试
  - 断言位置为定向测试目标
  - 记录断言失败与触发输入，生成复现PoC用例

```
1  static int connect_handler(struct io_event *e) {
2    // ...
3    // Generated assertion to validate client ID
4    assert(client_id_length_validation(c));
5    snprintf(cc->client_id, MQTT_CLIENT_ID_LEN, "%s
         ", c->payload.client_id);
6  }
7  // Generated helper function
8  static int client_id_length_validation(struct
       mqtt_connect *c) {
9    size_t original_len = strlen((char *)c->payload
       .client_id);
10   return (original_len < MQTT_CLIENT_ID_LEN);
11 }
```

实验设计　数据资源

- 数据资源

Subject	Version	LoC	Protocol	Specification
Sol	373d8	4.4k	MQTT 3.1.1	OASIS MQTT 3.1.1
TinyMQTT	6226ad	11.5k	MQTT 3.1.1	OASIS MQTT 3.1.1
Mosquitto	849e0f	46.2k	MQTT 5.0	OASIS MQTT 5.0
libcoap	17c3fe	45.3k	CoAP	RFC 7252
FreeCoAP	3adc2e	26.6k	CoAP	RFC 7252
pure-ftpd	381857	22.2k	FTP	RFC 959
uFTP	646404	6.7k	FTP	RFC 959
TLSE	1af154	41.8k	TLS 1.3	RFC 8446
wolfSSL	7fb750	1456.3k	TLS 1.3	RFC 8446
Dnsmasq	2.91	33.4k	DHCPv6	RFC 8415
NDHS	4b2728	5.6k	DHCPv6	RFC 8415

- 对比方法
  - 不一致检测准确性
    - Cursor（2025）
  - 生成测试用例有效性
    - AFLNET（2020）
    - 首个协议状态感知的模糊测试方法
- 评价指标
  - 漏洞数量

实验结果　对比实验

- **RQ1：现实中协议实现的不合规错误检测**
  - **420条规则，181条不合规错误，158项漏洞**
    - **故意偏离规范以提供扩展功能**
    - **多重不一致指向同一根本问题**
    - **更新的标准草案中允许某些违规行为**

Subject	Rules	Inconsistencies			Non-compliance Bugs
		TP	FP	Precision	
Sol	83	39	2	95.1%	39
TinyMQTT	83	29	3	90.6%	27
Mosquitto	118	15	2	88.2%	4
libcoap	30	4	1	80.0%	2
FreeCoAP	30	2	0	100.0%	2
pure-ftpd	54	17	2	89.5%	13
uFTP	54	18	1	94.7%	15
TLSE	58	25	2	92.6%	25
wolfSSL	58	7	1	87.5%	7
Dnsmasq	77	12	2	85.7%	11
NDHS	77	13	1	92.9%	13
Total	**420**	**181**	**17**	**90.6%**	**158**

ID	Project	Category	Bug Description	New	Status
1	Dnsmasq	Parsing	Missing Multicast Destination Check in dhcp6_packet Allows Unicast Message Processing	Yes	Confirmed
2	Dnsmasq	State	Missing Rapid Commit Check in dhcp6_no_relay Allows Unauthorized Rebind Bindings	Yes	Confirmed
3	Dnsmasq	State	Unconditional Lease Creation in dhcp6_no_relay for Rebind Messages	Yes	Confirmed
4	Dnsmasq	Parsing	Missing Zero Link-Address Check in dhcp6_maybe_relay	Yes	Confirmed
5	Dnsmasq	Parsing	Incorrect Hop-Count Check in relay_upstream6 Allows HOP_COUNT_LIMIT Messages	Yes	Confirmed
6	Dnsmasq	State	Missing Interface-Id Option in relay_upstream6 for Unusable Link-Address	Yes	Confirmed
7	Dnsmasq	Parsing	Improper Interface-Id Option Inclusion in dhcp6_no_relay Non-Relay Messages	Yes	Confirmed
8	Dnsmasq	Error	Incorrect NoAddrsAvail Status Placement in dhcp6_no_relay Reply Messages	Yes	Confirmed
9	Dnsmasq	State	Missing Requested Options in dhcp6_no_relay Advertise Messages	Yes	Confirmed
10	Dnsmasq	State	Missing IA_PD Handling in check_ia and dhcp6_no_relay for Solicit Messages	Yes	Confirmed
11	Dnsmasq	Parsing	Missing Required Options in dhcp6_no_relay for Solicit Message ORO	Yes	Confirmed
12	NDHS	Parsing	Missing Unicast Destination Check in process_receive Allows Invalid Message Processing	Yes	Reported
13	NDHS	Error	Improper Reply Sending in process_receive for Invalid CONFIRM Conditions	Yes	Confirmed
14	NDHS	Parsing	Missing ORO Content Validation in process_receive for Required DHCPv6 Options	Yes	Confirmed
15	NDHS	State	Missing Information Refresh Time Option in process_receive for Information-request Replies	Yes	Confirmed
16	NDHS	State	Unrequested Rapid Commit Option in process_receive Responses	Yes	Reported
17	NDHS	Parsing	Missing IA Content Validation in process_receive for Rebind Messages	Yes	Fixed
...					

实验结果　对比实验

- RQ2：LLM引导的程序切片&不一致检测分析
  - ProtocolGuard在不一致检测上优于通用AI代码编辑器
    - Cursor使用关键字搜索相似的代码片段，无法捕获跨功能的数据依赖性和控制流关系

Project	Cursor (Claude 3.7)			Cursor (DeepSeek R1)			ProtocolGuard (DeepSeek R1)		
	TP/FP/FN	Precision	Recall	TP/FP/FN	Precision	Recall	TP/FP/FN	Precision	Recall
Sol	37/3/7	92.5%	84.1%	16/5/28	76.2%	36.4%	**39/2/5**	**95.1%**	**88.6%**
pure-ftpd	16/9/4	64.0%	80.0%	10/14/10	41.7%	50.0%	**17/2/3**	**89.5%**	**85.0%**
libcoap	**5/3/2**	62.5%	**71.4%**	4/7/3	36.4%	57.1%	4/1/3	**80.0%**	57.1%
TLSE	21/5/7	80.8%	75.0%	18/11/10	62.1%	64.3%	**25/2/3**	**92.6%**	**89.3%**
Average	20/5/5	71.7%	76.8%	12/37/13	49.3%	52.0%	**21/2/4**	**86.3%**	**81.3%**

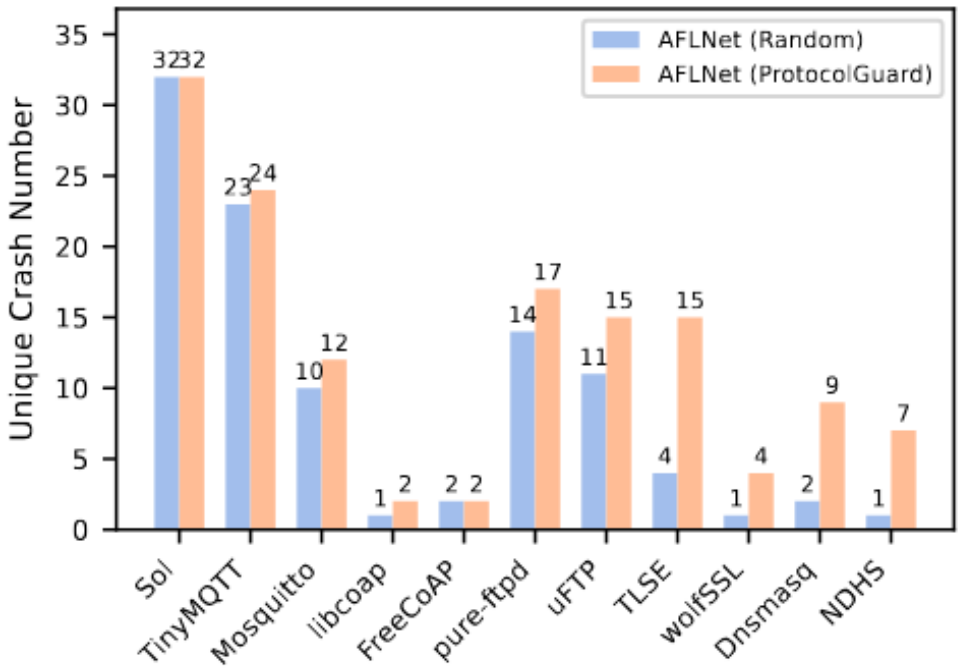
  - ProtocolGuard失败案例分析
    - 程序切片的完整性
      - Sol将消息处理与响应传输和连接处理分离，导致这些函数不与消息处理函数在同一调用路径上，程序切片不完整
    - LLM推理能力
      - Libcoap支持多种传输层代码，超出LLM上下文长度限制，降低LLM推理能力

实验结果　对比实验

- RQ3：断言生成的有效性检测
  - 语法正确性：可通过编译
  - 语义准确性：人工审查
    - 具有相应规则的约束
    - 收到不合规输入时能够终止程序
  - 实用性：定向模糊测试
    - 是否可由模糊器触发，从而导致程序崩溃

Protocol	Total	Syntactic	Semantic (Rate)	Crash (Rate)
Sol	41	41	39 (95.1%)	32 (78.0%)
TinyMQTT	32	32	27 (84.4%)	24 (75.0%)
Mosquitto	17	17	15 (88.2%)	12 (70.6%)
libcoap	5	5	4 (80.0%)	2 (40.0%)
FreeCoAP	2	2	2 (100.0%)	2 (100.0%)
pure-ftpd	19	19	19 (100.0%)	17 (89.5%)
uFTP	19	19	18 (94.7%)	15 (78.9%)
TLSE	27	27	22 (81.5%)	15 (55.6%)
wolfSSL	8	8	6 (75.0%)	4 (50.0%)
Dnsmasq	14	14	13 (92.9%)	9 (64.3%)
NDHS	14	14	12 (85.7%)	7 (50.0%)
Average	18	18	16 (88.9%)	13 (68.4%)

- RQ4：测试用例生成的有效性
  - ProtocolGuard 可以通过利用协议规范和已知的违规模式来生成高质量、语法有效且具有高度针对性的初始种子



算法总结

- 算法贡献
  - 提出一种结合LLM引导的静态分析与模糊测试的动态验证系统检测不合规错误
    - 利用LLM提取协议规则、程序切片，分析规则和代码之间的语义不一致
    - 根据不一致报告选用智能体生成断言与测试用例
    - 利用定向模糊测试触发断言失败，生成PoC
- 算法不足
  - 程序切片不准确
  - 断言生成的准确性依赖足量的上下文信息
    - 仅在静态代码上分析，无法访问程序的动态执行状态
  - 动态验证中应用AFLNET、SelectFuzz 等经典模糊器
    - 输入可能无法执行预期的逻辑路径，无法触发断言失败
  - 应用范围局限于C语言协议实现，C++具有面向对象的功能，静态分析更复杂

特点总结与未来展望

特点总结与未来展望

- 特点总结
  - 均利用智能体根据协议规范学习协议知识，检出逻辑缺陷（无显示崩溃）
  - BSFuzzer黑盒模糊测试方法，实际应用中无法获取源码
    - 智能体生成针对性测试用例并预测违规表现
    - 执行模糊测试，比较预期行为与实际响应是否存在不一致
  - ProtocolGuard灰盒模糊测试方法，可以获取源码
    - LLM提取规则的自然语言表述与对应规则的程序切片进行不一致检测
    - 利用智能体生成断言语句与测试用例
    - 定向模糊测试动态验证
- 未来展望
  - 智能体对未知协议/私有协议模糊测试的帮助?
    - 缺乏协议规范文档时要怎么处理

参考文献

[1] Zhang X, Zhang C, Li X, et al. A survey of protocol fuzzing[J]. ACM Computing Surveys, 2024, 57(2): 1-36.

[2] SONG X, PEI L, WU J, et al. ProtocolGuard: Detecting protocol non-compliance bugs via LLM-guided static analysis and dynamic verification. Proceedings 2026 Network and Distributed System Security Symposium[C]. Reston, VA: Internet Society, 2026.

[3] YANG T, QIN Y, ZHANG L, et al. BSFuzzer: Context-aware semantic fuzzing for BLE logic flaw detection. Proceedings 2026 Network and Distributed System Security Symposium[C]. Reston, VA: Internet Society, 2026.

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！