

IKEChecker: 语法引导的IKE协议有状态模糊测试

郑永辉, 赵栋梁, 顾纯祥, 张协力

(信息工程大学, 河南 郑州 450001)

摘要: 互联网密钥交换 (IKE, internet key exchange) 协议用于在IPSec (internet protocol security) 协议中进行身份验证和密钥协商, 其安全性对保护IP通信至关重要, 复杂的协议逻辑导致协议实现中难免存在安全漏洞。模糊测试是检测协议实现中潜在漏洞的有效手段, 但将现有模糊测试工具直接应用于IKE协议时, 存在生成测试用例质量不高、难以探索深层状态空间等局限性。针对上述问题, 设计了基于IKE协议语法的变异策略来减少无效测试用例的生成, 同时增加测试用例的多样性; 引入了基于进化策略的变异调度方案来自动优化变异算子的概率分布, 进一步提高生成高质量测试用例的概率; 设计了消息处理器用来维护协议交互上下文信息并执行密码学相关的操作, 从而支持在黑盒条件下对IKE协议进行测试, 并实现对深层协议交互行为和状态空间的探索。基于上述方法实现了IKE协议有状态模糊测试工具IKEChecker, 该工具支持对IKEv1和IKEv2协议进行测试。对两种广泛使用的支持IKE协议实现的开源软件strongSwan和Libreswan进行测试, 发现4个未公开漏洞; 通过与其他模糊测试工具对比, 评估了IKEChecker在漏洞检测方面的效率。

关键词: 安全协议; 模糊测试; 软件测试; IKE协议

中图分类号: TP393

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2024057

IKEChecker: grammar-guided stateful fuzzer for IKE protocol implementations

ZHENG Yonghui, ZHAO Dongliang, GU Chunxiang, ZHANG Xieli

Information Engineering University, Zhengzhou 450001, China

Abstract: The internet key exchange (IKE) protocol, integral to the authentication and key negotiation process within the Internet Protocol Security (IPSec) framework, has been utilized for the protection of IP communications. Given the complex protocol logic, security vulnerabilities in the implementation of the IKE protocol are inevitably present. Fuzz testing, recognized as an effective means of detecting potential vulnerabilities in protocol implementations, has been conventionally applied. However, the direct application of existing fuzzing tools to the IKE protocol has been found to present limitations, such as the generation of low-quality test cases and difficulty in exploring

收稿日期: 2023-12-28; 修回日期: 2024-07-26

通信作者: 郑永辉, yonghui.zh@163.com

引用格式: 郑永辉, 赵栋梁, 顾纯祥, 等. IKEChecker: 语法引导的IKE协议有状态模糊测试[J]. 网络与信息安全学报, 2024, 10(4): 109-122.

Citation Format: ZHENG Y H, ZHAO D L, GU C X, et al. IKEChecker: grammar-guided stateful fuzzer for IKE protocol implementations[J]. Chinese Journal of Network and Information Security, 2024, 10(4): 109-122.

deep states. To address these issues, a mutation strategy based on the grammar of the IKE protocol was designed, aiming to reduce the generation of invalid test cases while increasing the diversity of generated test cases. Additionally, an evolutionary strategy-based mutation scheduling scheme was introduced, which automatically optimized the probability distribution of mutation operators, further increasing the likelihood of generating high-quality test cases. A message handler was designed to maintain protocol interaction context information and perform cryptographic operations, thereby supporting testing of the IKE protocol under black-box conditions. This enabled the exploration of deep protocol interaction behavior and state space. Utilizing the aforementioned methods, an IKE protocol stateful fuzz testing tool named IKEChecker was implemented, supporting testing of both IKEv1 and IKEv2 protocols. Testing was conducted on two widely used open-source IKE protocol implementations, strongSwan and Libreswan, resulting in the revelation of 4 undisclosed vulnerabilities. By comparing IKEChecker with other fuzz testing tools, its efficiency in vulnerability detection was evaluated.

Keywords: security protocol, fuzz testing, software testing, IKE protocol

0 引言

随着互联网的快速发展,网络安全已成为人们关注的焦点。IPSec (internet protocol security) 作为一种保护网络数据传输的安全协议,广泛应用于虚拟专网网络,为IP通信提供机密性、完整性和身份认证服务,其安全性主要依赖于密钥管理和密钥交换。互联网密钥交换 (IKE, internet key exchange) 协议是IPSec协议的核心组成部分,负责协商、建立、修改和删除安全关联 (SA, security association),以实现动态的密钥管理。

IKE协议复杂性较高,在将协议规范转化为实际的软件和硬件实现过程中,可能会受到诸多不利因素的影响,包括但不限于编程错误、对规范的误解、性能优化导致的安全妥协,以及对第三方库和依赖的不当处理,从而导致协议实现的安全性无法满足协议规范所设定的安全目标,使系统容易遭受攻击。因此,对IKE协议实现进行深入的安全性分析具有至关重要的意义,旨在发现并修复潜在的安全问题,从而确保协议实现能够满足预期的安全需求,为网络通信提供可靠的保障。

模糊测试是一种应用广泛且高效的漏洞分析手段,按照对源码的访问需求大致可以分为白盒测试、黑盒测试和灰盒测试3类^[1]。黑盒测试指在不了解测试目标内部实现细节的情况下,通过输入一些测试数据,观察被测系统 (SUT, system under test) 的输出结果是否符合预期。白盒测试需要访问程序代码,基于对被测系统内部结

构和逻辑的了解来设计和执行测试用例。灰盒测试是介于黑盒测试和白盒测试之间的一种测试方法,需要对目标的内部实现有一定程度的了解,同时还会关注其外部功能是否符合要求。

路径空间爆炸问题导致白盒测试难以应用于复杂的安全协议实现。黑盒协议模糊测试工具,如Peach和Boofuzz等,通常需要预定义的协议模型来指导测试用例的生成。尽管黑盒测试生成的测试用例通常符合基本的协议格式,但缺乏有效的反馈引导机制,导致测试盲目进行,测试效果高度依赖于预定义模型的质量,代码覆盖率和漏洞发现效率较低。

近年来,AFL (american fuzzy lop) 的流行和应用证明了灰盒模糊测试的出色性能,随后出现了许多专用于协议的有状态灰盒模糊测试工具,如AFLNet^[2]和SGFuzz^[3]等。这些工具采用代码覆盖率和状态空间覆盖率作为反馈,基于遗传算法思想引导模糊测试过程向更有利的方向进行,在协议模糊测试中表现出色。然而,将这些通用的协议模糊测试工具应用于像IKE这样复杂的安全协议时,仍然存在以下局限性:①缺乏协议语法指导,变异随机性过强,导致生成大量无效的测试用例,同时难以生成具有某些特定结构(如特定载荷或字段)的测试用例;②随机等概率地选择变异操作,不考虑测试过程中不同的变异操作带来的收益差异,造成资源浪费;③无法处理加密和解密、完整性校验等密码学逻辑,导致难以有效探索深层协议状态空间。

为了解决上述问题,本文针对IKE协议设计

了一种基于IKE协议语法变异策略。该策略包含多种不同级别的变异算子,可以生成符合语法规则的消息输入,并保持一定随机性,以生成可能触发各种边界条件的测试用例。同时,引入了一种基于进化策略的变异调度方法,可以自动优化变异算子的概率分布,从而将更多能量分配给更有效的变异算子。此外,设计了IKE协议消息处理器,能够记录协议交互过程中的相关信息并执行密码学相关的操作。具体来说,本文的主要贡献如下:

1) 设计了一种基于IKE协议语法的变异策略以提高变异生成的测试用例质量,同时引入基于进化策略的变异调度进一步提高变异效率;设计了消息处理器用于维护协议交互上下文并执行密码学相关的操作,从而实现对深层协议状态空间的探索。

2) 基于上述方法对AFLNet进行扩展,实现了IKE协议模糊测试工具IKEChecker,支持在黑盒或灰盒条件下对IKEv1协议和IKEv2协议实现进行测试。

3) 对两款广泛应用的支持IKE协议实现的开源软件进行了测试,发现了4个新的漏洞;与其他几个测试工具进行对比,并对实验结果进行了分析。

1 相关工作

模糊测试的起源可以追溯到20世纪80年代末^[4],经过30多年的发展已成为漏洞挖掘领域最有效的手段之一。与基于文件的模糊测试相比,协议模糊测试面临着更多的挑战,受到学术界和工业界的广泛关注,本章主要介绍当前黑盒协议模糊测试和灰盒协议模糊测试的研究进展。

1.1 黑盒协议模糊测试

2001年奥卢大学开发的PROTOS^[5]可以看作协议模糊测试的起点,它利用定义的原语将协议中特定字段替换为默认初始值或特定的边界值。此后相关领域研究者陆续设计了多个黑盒协议模糊测试框架,其中最具代表性的有Peach、Boofuzz等。

Peach是一个扩展性良好、高度定制化的开源黑盒模糊测试框架,测试人员可以通过XML格式的Peach Pit文件来定义包括网络协议在内的

各种数据模型。Boofuzz是另一个广泛使用的开源黑盒协议模糊测试框架。它是经典协议模糊测试框架Sulley的继承者,完全基于Python实现,为用户提供了大量的数据格式(原语)来描述协议格式,并以有向图的形式描述协议会话流程。

这些传统的黑盒协议模糊测试框架基于预定义协议模型生成测试序列,其测试效果高度依赖于预定义模型的质量,且缺乏有效的反馈信息和引导机制,盲目随机地执行变异操作,从而影响测试效率。在最近的研究中,Bleem^[6]通过系统输出收集反馈信息,构建系统状态跟踪图指导测试数据包序列生成,为黑盒协议模糊测试引入了有效的反馈机制。此外,其采用中间人的工作模式,通过修改协议双方交互的数据包以实现变异,从而避免手动定义数据格式和协议模型。然而该工作方式限制了其在具有复杂密码学逻辑的安全协议上的应用。

1.2 灰盒协议模糊测试

2014年,由谷歌安全工程师开发的灰盒模糊测试工具AFL问世,相比早期的黑盒模糊测试工具,其引入了遗传算法思想,以代码覆盖为导向,保存触发新的代码执行路径的测试用例作为种子用于后续测试。考虑到协议实现通常具有复杂的状态空间,Pham等^[2]在AFL的基础上提出了首个有状态的灰盒协议模糊测试工具AFLNet。它使用服务器的响应码来识别服务器的状态,并通过状态反馈来指导模糊测试过程,与Boofuzz等黑盒协议测试工具相比,其在代码覆盖和漏洞发现方面表现更佳。

为了更精确地识别网络服务的状态,在AFLNet的基础上,StateAFL^[7]使用细粒度的长生命周期变量表示程序的状态。SGFuzz则通过静态分析提取所有枚举变量并过滤出可能的状态变量来标定状态。为提升AFLNet测试吞吐量,SnapFuzz^[8]将AFLNet中的异步网络通信转换为基于Unix套接字通信以提高通信速度。SNPS-Fuzzer^[9]引入了进程快照技术,可直接使被测系统恢复到目标状态。NSFuzz^[10]将网络事件循环入口作为与模糊器之间的同步点,减少模糊器发送下一条消息的等待时间。在种子调度策略方面,AFLNet_Legion^[11]将AFLNet的响应状态码

序列建模为蒙特卡洛树，并采用 Legion^[12] 算法选择目标树节点及对应的种子，但并没有表现出明显优于 AFLNet 的性能。

在变异策略方面，基于 AFL 的协议模糊工具则基本沿用 AFL 的随机变异策略，缺少协议语法指导，无法感知特定的协议消息格式，从而影响通过变异得到的测试用例的质量：一方面，过强的随机性可能会严重破坏消息格式，导致其过早地被丢弃；另一方面，无法识别特定的数据结构，导致生成的测试用例的多样性有限。

2 IKE 协议基础知识

2.1 IKE 协议消息格式

IKE 协议目前有 IKEv1^[13] 和 IKEv2^[14] 两个版本，两者都是基于 ISAKMP^[15] 协议设计的。ISAKMP 是一个框架协议，提供了一种定义和交换安全策略的机制，以及在互联网环境中建立、协商、修改和删除安全关联的通用框架。IKEv1 和 IKEv2 协议消息格式与互联网安全连接和密钥管理协议（internet security association and key management protocol, ISAKMP）基本一致，只有细微差异，简单起见，这里只介绍 ISAKMP 协议消息的基本格式。

如图 1 所示，一个完整的 ISAKMP 协议消息由一个消息头（header）和多个载荷（payload）组成。消息头由 8 个固定长度的具有不同含义的字段组成。载荷有多种类型，其通用格式如图 2 所示，包含 next payload、reserved 和 payload length 3 个基本字段，并携带不同的载荷数据。不同类型载荷可能会定义多个具有不同含义和功能的其他字段，每种载荷的具体格式在 IKE 协议相关的 RFC 文档中都有详细的定义，在此不再赘述。

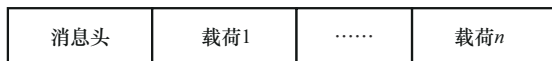


图 1 ISAKMP 协议消息格式

Figure 1 ISAKMP protocol message format

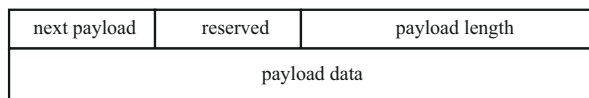


图 2 ISKAMP 协议载荷通用格式

Figure 2 ISAKMP protocol payload common format

2.2 IKE 协议交互流程

图 3 描述了 IKEv2 协议建立 IKE SA 的基本流程。在初始交换中，SAi1 载荷声明发起方支持的密码套件，KEi 载荷携带发起方的 DH（Diffie-Hellman）公钥信息，Ni 载荷则提供了随机值。响应方从发起方提供的密码套件中选择支持的配置，并在 SAr1 载荷中表达该选择，KEr 与 Nr 载荷的功能与发起方一致。初始交互完成后，发起方与响应方分别根据协商的算法和交换的数据计算密钥材料，此后的消息都应受到加密和完整性保护。在随后的认证交互中，双方通过发送携带身份信息的 ID 载荷及包含认证数据的 AUTH 载荷验证对方身份并建立 IKE SA。

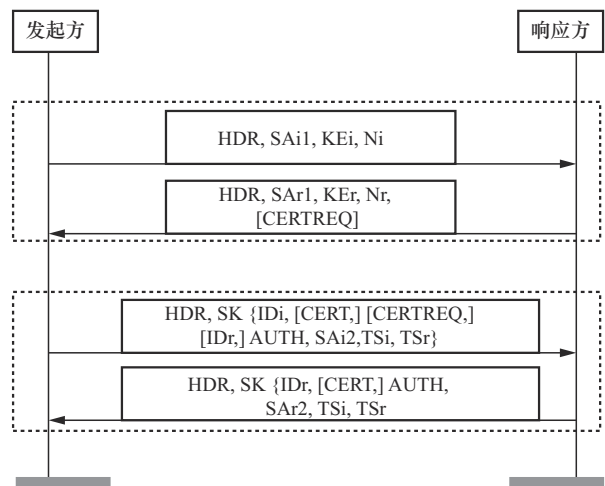


图 3 IKEv2 协议建立 IKE SA 流程

Figure 3 Process for establishing IKE SA in IKEv2 protocol

上述流程只是一个简单的 IKE SA 建立流程，除此之外，在协议标准文档中还定义了各种场景下的协商流程及各种各样的异常处理机制。所有这些正常和异常的情况构成了协议逻辑规范，也称为协议状态机。安全协议实现中的具体逻辑并不总是与协议规范完全一致，这导致在特定的协议实现状态机中可能存在安全风险，要发现这些异常逻辑，需要对协议状态空间进行有效探索。

3 IKEChecker 设计

本章介绍 IKEChecker 的整体设计框架和其主要模块的实现细节。IKEChecker 在协议模糊测试框架 AFLNet 的基础上进行了扩展，图 4 描述

了其基本工作流程。IKEChecker被设计为支持在黑盒或灰盒条件下对IKE协议进行模糊测试,前者无须对源码或二进制文件插桩,从测试过程中生成的状态机中提取反馈信息,后者则同时考虑将代码覆盖作为另一个重要的反馈指标。

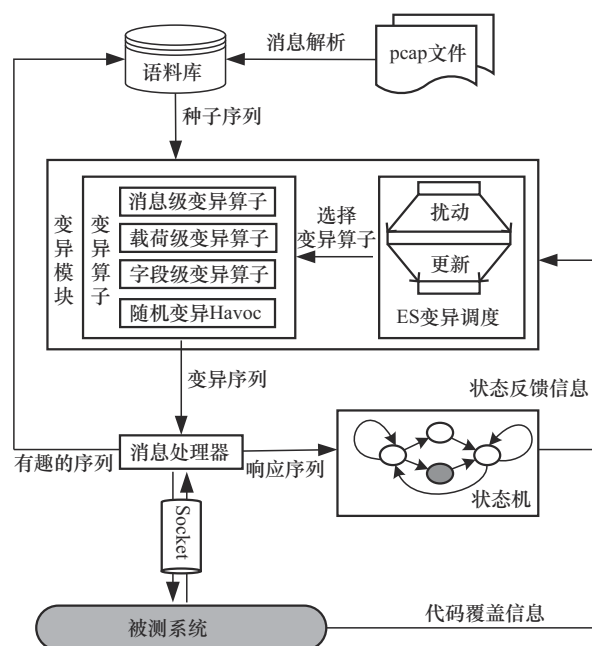


图4 IKEChecker的基本工作流程
Figure 4 Basic workflow of IKEChecker

IKEChecker解析捕获的IKE协议交互数据作为初始语料库,对于加密的消息,IKEChecker将其解析为明文进行保存。对于选中的种子序列,IKEChecker采用基于IKE协议语法的变异策略来生成高质量的测试用例,所有变异操作都在明文格式的IKE消息上执行,与密码学相关的操作均交由消息处理器模块进行处理。消息处理器模拟协议交互的一端,记录协议交互行为与当前IKE会话相关的信息,计算或更新密钥材料,从而能够对变异生成的消息序列执行密码学相关的操作。消息处理器通过Socket接口与被测系统进行交互,解析响应消息并将其表示为抽象符号,随后将抽象响应序列反馈至状态机模块以更新状态机信息。如果发现新的状态或状态转移路径(或者在灰盒条件下,发现新的代码执行路径)则认为当前测试序列是有趣的,将其添加到语料库中,同时更新当前变异算子的适应度值,以实现基于进化策略的变异调度方案。

3.1 基于IKE协议语法的变异策略

尽管字节级的随机变异(如AFL中的Havoc)在简单的软件程序测试中取得了良好的效果^[16],但将其应用于IKE协议这样具有复杂格式和严格验证规则的安全协议时,测试效率显著降低,原因有以下几点:①随机变异可能使消息格式遭到严重破坏而过早地被测试目标丢弃,从而产生大量无效的测试用例;②随机变异无法识别特定结构,只能在字节级别上进行变异,不能操作更高级别的数据结构,导致变异结果多样性有限;③安全协议有复杂的验证机制,如使用随机数来抵抗重放攻击,通过加密和完整性验证以确保机密性和完整性,随机变异无法通过这些验证。

为解决上述①和②问题,IKEChecker实现了基于IKE协议语法的变异策略。协议语法规则规定了协议消息的具体格式和字段含义,对协议消息进行变异时,IKEChecker会按照协议语法对其进行解析,然后执行载荷级或字段级的变异操作。基于协议语法进行变异,一方面可以使变异后的消息仍保持正确的格式,从而降低生成无效测试用例的概率;另一方面可以实现比字节更高级别的结构变异,从而使测试用例更具多样性。

除了支持单个消息的变异外,IKEChecker还支持另外两个级别的变异操作。一是通过插入或删除消息等操作对协议消息序列进行变异;二是保留了字节级的随机变异。表1列出了该变异策略包含的各种变异算子。

1) 消息级变异。为了有效地探索协议的状态空间,IKEChecker执行消息级的变异操作,以生成各种不符合协议规范定义的交互流程的输入序列。引入这种变异操作是受模型学习^[17]的启发,模型学习在L*[18]等学习算法的驱动下执行不同的输入序列来推断协议状态机,然后查找协议实现中可能出现的错误逻辑。模糊测试的目的是在短时间内找到尽可能多的漏洞,而非推导一个完整的状态机模型,因此IKEChecker不使用学习算法来驱动测试用例生成,只是随机地执行变异。

2) 载荷级变异。载荷是IKE协议消息中的一个基本结构,在协议规范定义的交互过程中,

表 1 IKEChecker 变异策略包含的变异算子

Table 1 Mutators contained in the IKEChecker mutation strategy

变异级别	变异对象	变异算子
消息级	IKE 协议消息序列	从其他种子中随机选择一个消息替换当前序列中的多个消息； 从其他种子中随机选择一个消息随机插入当前序列中； 随机重复当前序列中的一个消息； 随机删除当前序列中的一个消息
载荷级	IKE 协议消息	随机重复当前消息中的一个载荷； 随机删除当前消息中的一个载荷； 随机从其他消息中选择一个载荷并插入当前消息中； 随机生成一个新的载荷并插入当前消息
字段级	IKE 协议消息	随机从当前消息的头部中选择一个字段进行变异； 随机从当前消息中选中一个载荷并从中随机选择一个字段进行变异
字节级	IKE 协议消息序列原始数据	AFL 中 Havoc 阶段的各种变异算子, 如比特翻转、随机字节替换等

指示了每个消息应该携带的载荷类型。冗余或缺失的载荷可能会导致被测目标出现异常行为。除了随机插入、删除和重复某个载荷外, IKE-Checker 还会随机生成某个载荷并将其插入当前消息中。这种操作的考虑是, IKE 协议消息有多种载荷类型, 而初始语料库中的消息序列可能无法覆盖所有类型。

3) 字段级变异。大多数字段只有很少的有效值, 如果随机设置字段值会扩大探索空间、降低测试效率, 甚至会完全错过真正有趣的值。例如, 长度为 1 字节的枚举类型字段可能只包含几个有效值, 但随机变异会探索 256 个可能值。为避免在较大的无效值空间中进行搜索, IKE-Checker 参考 RFC 文档来确定目标字段的取值范围。

4) 字节级变异。上述级别变异操作生成的是基本符合语法约束的测试用例, 考虑到实际测试中某些漏洞可能由错误格式的消息触发, IKE-Checker 保留了 AFL 中 Havoc 阶段采用的部分变异操作。与完全随机的变异策略相比, 该策略保持了一定的随机性, 且不会导致过多的资源被浪费。

在实际应用中, 以 AFL 为代表的灰盒模糊测试工具通常会采用“堆叠”(stack)的策略, 即对种子应用多个变异算子后才生成一个测试用例。实际上, 在协议模糊测试中堆叠策略的应用更加重要, 因为对种子进行变异的开销远

小于执行一个测试序列(向目标服务器发送消息并接收响应)的开销。AFL 的堆叠策略是每次从所有变异算子中随机选择一个执行。IKE-Checker 的变异策略设置了不同级别的变异算子, 需要对种子序列进行不同粒度的解析(例如, 消息级变异只需要将种子切割为多个消息即可, 而字段级变异需要精确解析到字段粒度), 因此在整个堆叠周期中, IKEChecker 只选择某一个级别的变异算子。

3.2 基于进化策略的变异调度

IKEChecker 的变异策略中定义了多个变异算子, 一个自然的问题是在测试中如何选择变异算子以实现最大收益。现有方法大多是随机等概率选择, 但对不同目标进行测试时, 变异算子带来的收益通常是不同的, 甚至会有较大差异。因此有必要设计一种自适应方法, 自动调节变异算子的概率分布, 以此来提高测试用例的效率。这可以看作是一个优化问题, 候选解可以定义为不同变异算子被选中的概率, 优化目标则是通过变异生成更有效的测试用例。

由于这个优化问题没有明确的目标函数, 因此不适用需要梯度信息的经典优化算法。在最近的研究中, DARWIN^[19]利用进化策略(ES, evolution strategies)优化 AFL 中变异算子的概率分布, 从而取得了优于 AFL 的表现。进化策略是一种模仿生物进化过程的优化方法, 每一代的解(个体)都通过变异产生后代, 然后根据某种

选择策略选择最优的个体进入下一代。其基本步骤如下。

1) 初始化: 随机生成一个初始解的集合, 每个解都是优化问题的一个可行解。

2) 变异: 对种群中的每一个个体, 通过随机扰动 (例如高斯噪声) 来产生一个或多个新个体。

3) 评估: 计算所有个体 (包括父代和后代) 的适应度值。适应度值根据优化问题来定义, 反映一个个体的质量。

4) 选择: 从所有个体中选择适应度值最高的一部分个体进入下一次迭代。

5) 终止: 如果满足终止条件 (如达到最大迭代次数, 或者最优解的质量已经达到要求), 则算法结束, 输出当前最优的解; 否则, 返回第2)步, 进行下一轮的变异和选择。

在 DARWIN 变异调度方案中, 整个堆叠循环中只应用某一个变异算子, 由于 IKEChecker 采取分层级的变异策略, 这种方式是不适用的。(考虑下面这种情况, IKEChecker 选中的变异算子为“随机删除当前序列中的某个消息”, 应用堆叠策略后, 最坏的结果是得到一个空序列, 这样的变异显然是低效的)。因此, 正如 3.1 节所述, IKEChecker 在一次堆叠循环中应用一个级别的变异算子。

进化策略候选解的编码形式有两种: 一种是使用实值向量表示候选解, 向量中每个元素表示对应的变异算子被选中的概率大小; 另一种是使用二进制向量表示候选解, 即向量值只取 0 或 1, 每次随机选择值为 1 的元素对应的变异算子。对 IKEChecker 而言, 由于只有 4 个级别的变异操作, 即候选解向量维度为 4, 这种情况下使用实值向量比二进制向量能提供更准确的概率控制。

协议实现通常采用的是具有复杂状态空间的有状态系统, 系统在不同的状态下会产生不同行为, 某些异常行为只有在特定状态下才能触发。因此在协议模糊测试中, 除代码覆盖率外, 另一个需要关注的重点是状态覆盖。IKEChecker 中将进化策略的适应度值表示为:

$$\text{fitness} = \text{paths} + \text{trans} \quad (1)$$

其中, paths 表示发现的代码执行路径数量, 用

于衡量代码覆盖; trans 表示发现的状态转移路径数量, 用于衡量状态覆盖。黑盒条件下由于无法获取代码执行路径信息, 因此仅使用 trans 评估适应度值。

算法 1 给出了 IKEChecker 对一个种子进行变异时采用变异调度的基本流程。首先随机生成 μ 个父代候选解 (line1), 对于每个父代通过高斯扰动生成 λ 个子代候选解 (line4), 之后对每个子代进行评估: 根据子代确定变异级别 (line6), 对种子应用该级别变异所包含的各种变异算子以生成测试用例 (line7), 执行测试用例并根据目标反馈评估适应度 (line8)。根据适应度确定表现最佳的子代 (line9~12), 如果优于当前父代, 则选取该子代作为下一轮迭代的父代 (line14~16)。

算法 1 变异调度流程

输入 seed //待变异种子;

μ //进化策略父代数;

λ //每个父代的子代数

输出 testCase //变异生成的测试用例

```

1) parents ← initializeParents( $\mu$ )
2) repeat
3)   for each parent do
4)     children ← perturbation(parent,  $\lambda$ )
5)     for each child do
6)       mutateLev ← chooseMutateLevel(child)
7)       testCase ← Mutate(mutateLev, seed)
8)       fitness ← execTestCsaec(testCase)
9)       if fitness > bestfitness then
10)        bestfitness ← fitness
11)        bestChild ← child
12)      end if
13)    end for
14)  if bestfitness > parentfitness then
15)    parents ← bestChild
16)  end if
17) end for
18) until 测试中止

```

3.3 消息处理器

安全协议有复杂的验证机制, 如果将变异后的消息序列直接发送给被测系统, 那么这些消息

序列无法通过验证机制。以 IKEv2 协议为例, 当协议双方完成初始交换后, 接下来, 所有消息都应该受到加密和完整性保护。如果不能对变异后的消息执行正确的密码学操作, 那么测试过程只能局限于浅层交互行为, 无法深入探索协议的状态空间。因此, 在 IKEChecker 中设计了消息处理器用来解决这个问题。

如前所述, 变异模块只负责对明文消息进行变异, 变异结果交由消息处理器进行进一步处理。对于变异序列中的每个消息, 消息处理器根据当前的协议交互上下文信息对其进行适配。

消息处理器可以看作一个简单的客户端, 模拟协议交互的发起方, 维护当前的 IKE 会话 SPI 及 Message ID, 使发送至被测系统的消息通过最基本的验证。更重要的是, 消息处理器记录协议交互的数据并从中提取重要信息, 用于计算和更新当前会话的密钥材料, 从而实现对变异消息的正确加密及对被测系统响应消息的正确解析。

具体来说, IKE 协议中 SA 载荷用于协商密码套件, KE 载荷用于交换 DH 公钥信息, NONCE 载荷用于交换随机值, 协议双方通过这些载荷中的关键信息计算密钥材料。消息处理器在发送消息前会对消息进行解析, 提取上述特定载荷中的关键信息。消息处理器若接收到带有与发送的载荷类型相匹配的载荷消息, 则会提取其中的关键信息, 并据此更新密码参数并重新计算密钥材料。例如, 如果发送的消息中带有 KE 载荷且收到的响应消息中也有 KE 载荷, 消息处理器将提取其中的公钥来重新导出密钥。消息处理器根据最新的密钥材料对变异后的明文消息执行加密操作, 以及对被测系统响应消息执行解密操作。

由于变异固有的随机性, 消息处理器可能会在解析变异消息过程中遇到错误, 这时消息处理器会放弃提取关键信息。这种消息通常也无法被测系统正确解析, 因此无须更新密钥材料。同样, 如果解密失败, 消息处理器放弃对响应消息的进一步解析和信息提取。

为通过 IKE 协议中的身份认证机制, 消息处理器还需能够计算正确的认证信息。认证信息的计算除需正确的密钥外, 还涉及之前的某些交互数据 (例如, IKEv2 计算 AUTH 载荷时, 签名数

据 InitiatorSignedOctets 中就包含了完整的初始交换消息)。与计算密钥材料类似, 消息处理器使用所有最新的数据来计算认证信息。由于模糊测试过程中会出现各种不确定情况, 消息处理器无法保证构造认证载荷的逻辑与被测目标一致, 但至少能够在交互流程完全正常的情况下通过认证, 从而可以在深层状态空间中进行测试。

3.4 IKEChecker 的状态表示

AFLNet 用一个整型变量来抽象表示不同的响应消息, 因为类似文件传输协议 (FTP, file transfer protocol) 这样的协议消息中存在一个响应码用于表示协议消息的含义。但在 IKE 协议中不存在这样的响应码, IKE 协议消息由消息头部和多个有效载荷构成, 因此 IKEChecker 使用一个字符串来对一个协议消息进行抽象表示, 具体表示方式如下:

$$V_Etype^{[*]1-Ptype_1-\cdots-Ptype_n} \quad (2)$$

其中 V 表示 IKE 协议消息的版本; $Etype$ 表示 IKE 协议消息的交换类型, $*$ 的有无表示该消息是否加密; $Ptype_i$ 则表示 IKE 协议消息中第 i 个载荷所对应的载荷类型。需要注意的是, IKE 协议消息中同类型的载荷表示的含义是类似的, 只是具体值有所差异, 但通知载荷是一个例外, 通知载荷中包含通知类型字段, 该字段指示通知载荷的真实含义。因此对于通知载荷, $Ptype_i$ 表示载荷中的通知类型。例如, 在对 IPsec 协议开源软件 strongSwan 的 IKEv2 模块进行测试时, 在初始交换中接收到的响应消息可抽象表示为:

$$v2_SAINIT-SA-KE-Ni-4022-4014 \quad (3)$$

其中, 4022 和 4014 为两个通知类型 CHILD_LESS_IKEv2_SUPPORTED 和 MULTIPLE_AUTH_SUPPORTED 对应的十六进制取值。

4 实验评估

本节从以下 3 个方面对 IKEChecker 进行综合评估。

1) IKEChecker 能否发现真实 IKE 协议实现中未公开的漏洞。

2) IKEChecker 的基于 IKE 协议语法的变异策略和基于进化策略的调度算法的有效性。

3) IKEChecker 与现有的 IKE 协议模糊测试工具相比在漏洞发现方面的效率。

下面详细描述实验部分的测试对象和实验设置,并对实验结果进行分析。

4.1 测试对象与实验环境

实验环节选择 strongSwan 和 Libreswan 这两种软件中普遍采用的 IPSec 协议实现作为测试对象。strongSwan 和 Libreswan 支持各种版本和各种配置的 IKE 协议,满足对 IKE 协议进行模糊测试的需求,且两者都是开源软件,具有可修改和定制的优势,便于在测试过程中根据需要进行必要的修改和定制。

实验环境为安装了 Ubuntu 20.04.4 LTS 的虚拟机。虚拟机的硬件配置为 8 核 Intel i9-11900 处理器与 8 GB 内存。为了消除模糊测试实验的不确定性,所有实验均在具有相同配置的虚拟机上进行。

4.2 漏洞检测能力评估

使用 IKEChecker 对 strongSwan 和 Libreswan 进行测试,发现了 4 个未公开漏洞,其中 3 个已经分配了通用漏洞披露 (CVE, common vulnerabilities and exposures) 编号,下面对这 4 个漏洞分别进行简单描述。

1) Libreswan-vul1 (CVE-2023-23009): 该漏洞存在于特定版本的 Libreswan IKEv2 协议实现中,Libreswan 处理 traffic selector 载荷时,没有检查函数返回值,因此通过一个异常的 traffic selector 可能会触发一个空指针引用,从而导致 Libreswan 的守护进程 pluto 崩溃并重启。要构造这个异常的 traffic selector 载荷,可以将 selector length 字段值设置为一个异常值,Libreswan 解析该载荷时会遇到错误而返回空指针。利用这个漏洞,远程攻击者可以进行拒绝服务攻击。Libreswan 只在对端通过认证之后才会处理 traffic selector 载荷。

2) Libreswan-vul2 (CVE-2023-30570): 该漏洞存在于特定版本的 Libreswan IKEv1 野蛮模式认证流程中。Libreswan 接收到带有不可接受的算法配置的野蛮模式消息时,会响应一个“NO_PROPOSAL_CHOSEN”类型的通知消息,其响应方 cookie 值非零。攻击者重放第一个野蛮模式消息,并将其响应方 cookie 设置为接收到的值,此消息会导致 Libreswan 守护进程 pluto 崩溃

并重启。利用此漏洞,攻击者可以在不进行身份验证的情况下执行拒绝服务攻击。只有当目标配置为 IKEv1 野蛮模式时,才能触发此漏洞。

3) Libreswan-vul3 (CVE-2023-38710): 该漏洞存在于特定版本的 Libreswan IKEv1 协议实现中,当双方的 ID 类型设置为 IPv4_ADDR 时,通过发送带有 ID_FQDN (0x02) 类型的 IDir 载荷的快速模式的数据包会导致 Libreswan 的守护进程 pluto 崩溃并重启。Libreswan 只有在通过主模式或野蛮模式认证后,才会处理快速模式消息。

4) strongSwan-vul1: 该漏洞存在于特定版本的 strongSwan IKEv1 协议实现中。该漏洞通过测试过程中生成的简易状态机发现,其中存在这样一条异常状态路径: v1_Main-SA-VID-VID-VID→v1_Mian-KE-NONCE→v1_Aggr_ID-Hash。在正常的 IKEv1 主模式交互流程中,响应方发送的第 3 条消息应当是一个携带 ID 载荷和 Hash 载荷的加密消息。如果将发起方发送的第 3 条消息中的 Exchange Type 字段由 0x02 (identity protection) 修改为 0x04 (aggressive mode),响应方发送的第 3 条消息将不会被加密。即使没有配置野蛮模式的情况下,上述现象仍会出现。虽然这个漏洞似乎不会带来严重的安全风险,但它违反了 identity protection 交换的要求。

4.3 变异策略和变异调度算法有效性评估

为了验证 IKEChecker 采用的变异策略和变异调度算法的有效性,本节比较了 IKEChecker 与 AFLNet_IKE 和 IKEChecker_rand 的性能。下面对 AFLNet_IKE 和 IKEChecker_rand 两个工具进行简单介绍。

AFLNet_IKE 扩展了 AFLNet, 添加了消息处理器使其支持对 IKEv1 和 IKEv2 协议实现进行测试。与 IKEChecker 相比,它没有采用基于 IKE 协议语法的变异策略和相应的变异调度算法,而是沿用了 AFLNet 中原始的 Havoc 变异策略。通过与该工具对比来评估基于 IKE 协议语法的变异策略的有效性。

IKEChecker_rand 在 AFLNet_IKE 的基础上引入了本文设计的基于 IKE 协议语法的变异策略,但不采用基于进化策略的变异调度算法,即

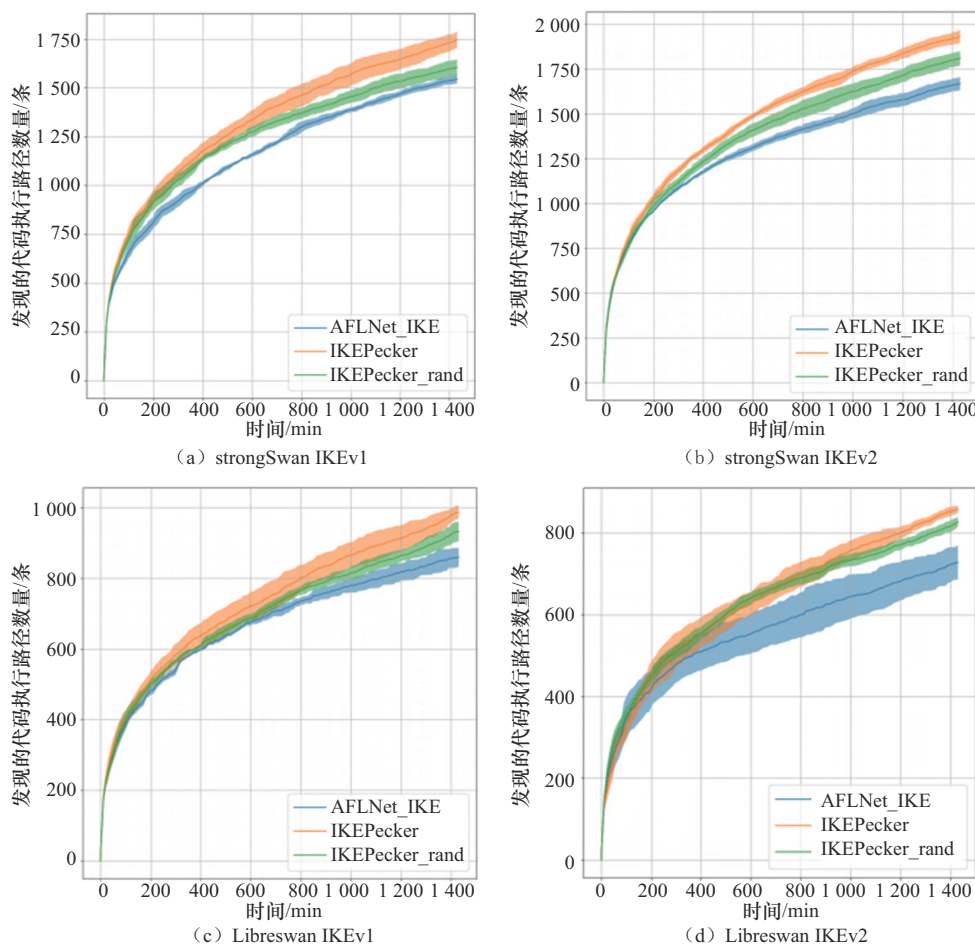


图5 使用IKEChecker、IKEChecker_rand和AFLNet_IKE测试发现的代码执行路径数量

Figure 5 Number of code execution paths discovered by IKEChecker, IKEChecker_rand and AFLNet_IKE testing

每次都随机选择不同的变异操作对种子进行变异。通过与该工具进行对比来评估基于进化策略的变异调度算法的有效性。

本节使用上述3种工具测试了最新版本的strongSwan和Libreswan, 包括其IKEv1协议和IKEv2协议。为减少实验误差, 使用每个工具对每个目标执行5轮测试, 每轮测试时长为24 h (将IKEChecker集成到profuzzbench^[20]中, 通过运行多个Docker容器进行实验)。

从3个方面对比IKEChecker、AFLNet_IKE和IKEChecker_rand的表现: 发现的代码执行路径数量、发现的状态数量和状态转换路径的数量。第一个指标从代码覆盖的角度衡量3种工具的表现, 后两个指标则是从状态覆盖的角度来评估。图5展示了所发现的代码执行路径数量随时间变化的趋势, 其中实线表示多轮实验结果的平均值, 阴影区域表示95%的置信区间。图6统计

了在每一轮测试中所发现的状态的数量。图7与图5相似, 但描述的是发现的状态转换路径数量随时间变化的趋势。可以看到, 这3种工具在上述3个指标上表现出了基本一致的结果, 即IKEChecker的表现优于IKEChecker_rand, IKEChecker_rand优于AFLNet_IKE。这表明了IKEChecker实现的基于IKE协议语法的变异策略和基于进化策略的变异调度算法的有效性。具体来说, 基于IKE协议语法的变异策略可以在特定结构(如特定载荷或字段)上进行变异, 使IKEChecker_rand生成的测试用例更具多样性, 同时可以减少大量无效测试用例的生成, 从而使其比AFLNet_IKE覆盖更多的代码路径和状态。变异调度算法基于发现的代码路径和状态路径的数量评估适应度值, 将更多的资源分配给能发现更多代码路径和状态路径的变异算子, 使IKEChecker的表现优于IKEChecker_rand。

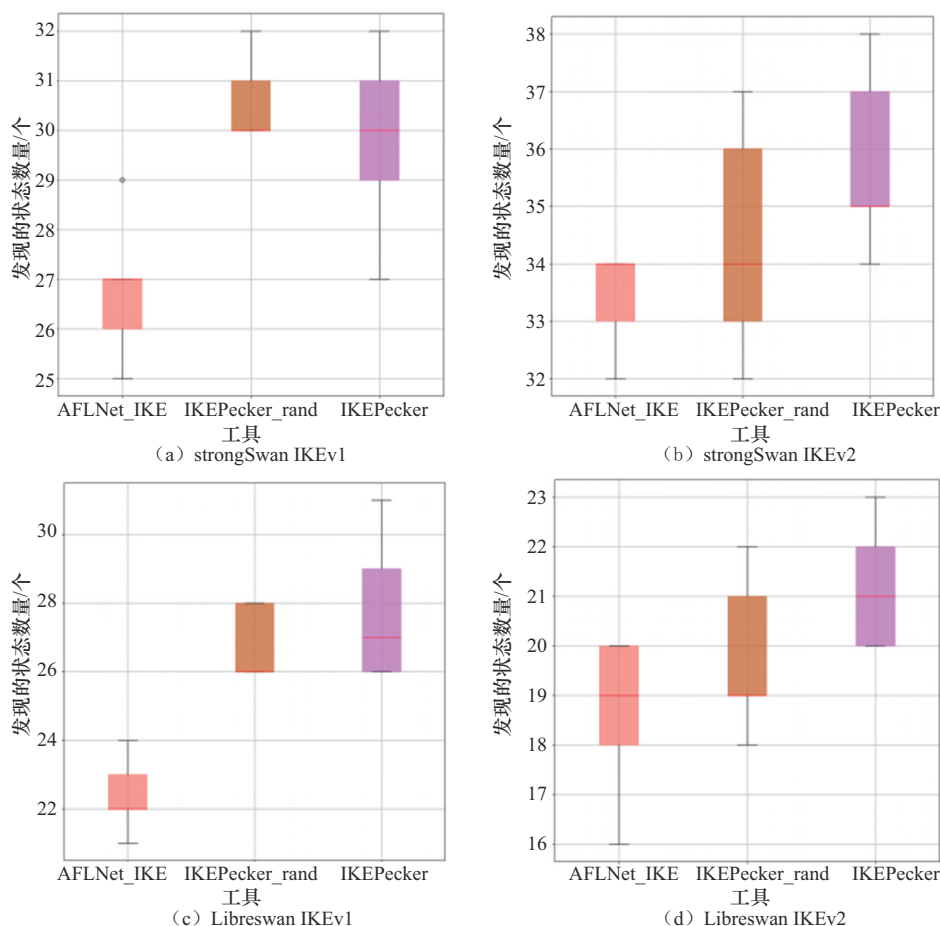


图6 使用IKEChecker、IKEChecker_rand和AFLNet_IKE测试发现的状态数量

Figure 6 Number of states discovered by IKEChecker, IKEChecker_rand and AFLNet_IKE testing

4.4 与其他测试工具对比

为比较IKEChecker和其它工具在漏洞发现方面的能力,整理了多个公开的Libreswan和strongSwan漏洞(包括CVE上的漏洞和4.2节最新发现的漏洞),用不同的工具对这些漏洞对应版本的软件进行测试,统计每种工具发现漏洞的数量和时间。除IKEChecker_rand和AFLNet_IKE外,还选择了下列3种模糊工具进行对比。

SECFuzz^[21]是一种专用于IPSec协议实现的轻量级模糊测试工具。其工作模式类似于Bl-eem,被设置为工作在两个正常的IPSec协议实现之间的中间人。它收集由客户端发往目标服务器的IKE消息,对其进行变异后,再将消息发送给服务器,同时使用动态内存分析工具检测目标服务器异常。SECFuzz选用Openswan作为客户端生成IKE消息,并从Openswan生成的日志中提取密钥信息,由于Openswan已经停止维护,

这里采用Libreswan替代Openswan,并对SEC-Fuzz中的相应代码进行了修改。

L*+nuXmv结合模型学习和模型检查来验证系统中的安全属性。具体来说,该方法基于模型学习方法构建反映目标系统执行逻辑的状态机模型,然后通过模型检查器nuXmv来验证模型是否满足给定的属性。该工具在2016 CAV大会^[22]上发表,随后相关研究者将其应用于SSH^[23]、IPSec^[24]等安全协议。这部分实验主要基于Github上的开源项目IPSecStateLearner,通过扩展使其兼具对IKEv1和IKEv2协议进行测试的能力。

SGFuzz是一种通用的有状态灰盒模糊测试工具,使用了与AFLNet不同的状态表示方式。SGFuzz首先通过静态分析提取程序中所有枚举变量,并基于特定规则过滤出可能的状态变量,在模糊测试过程中根据这些变量的分配构建状态

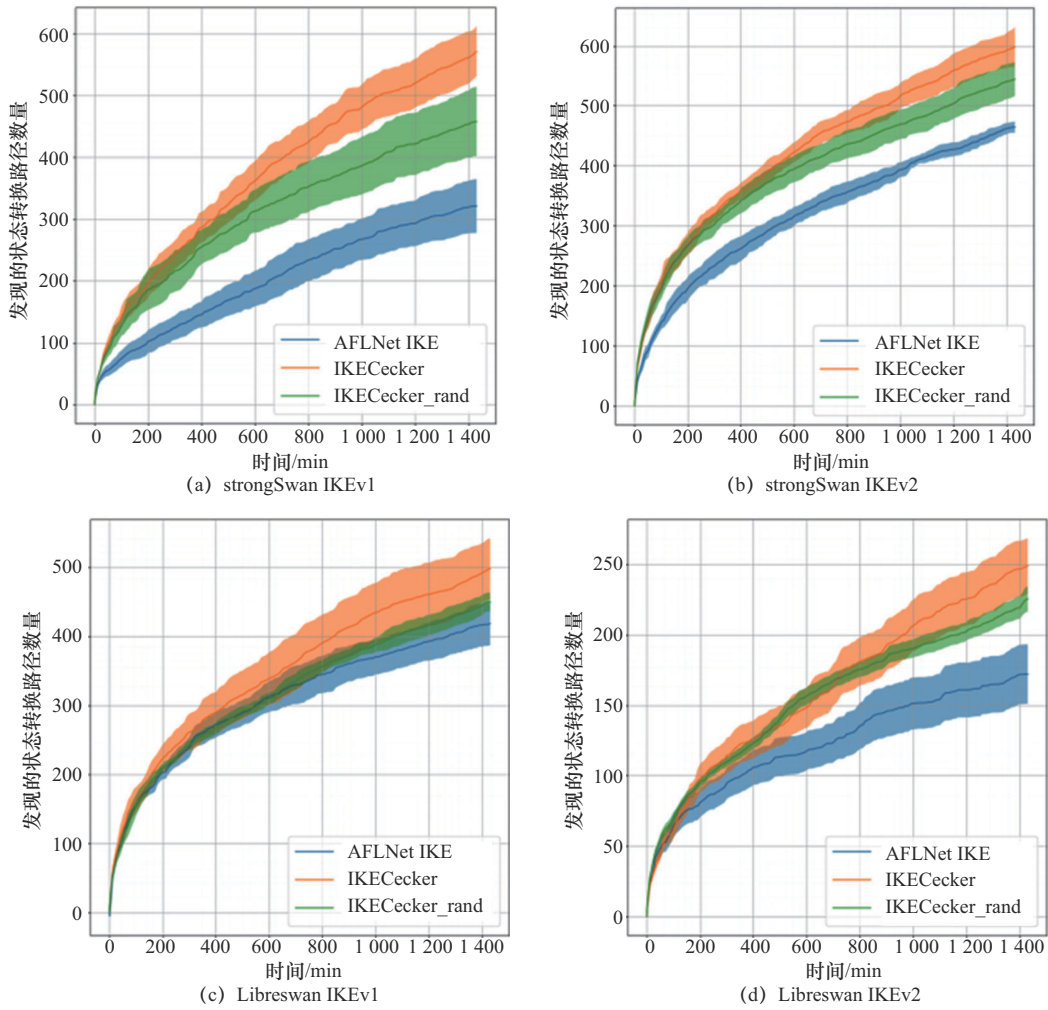


图7 使用IKEChecker、IKEChecker_rand和AFLNet_IKE测试发现的状态转换路径数量

Figure 7 Number of states transition paths discovered by IKEChecker, IKEChecker_rand and AFLNet_IKE testing

转移树，用于指导模糊测试更有效地探索协议状态空间。SGFuzz 基于进程内的、覆盖引导的模糊测试工具 LibFuzzer 实现，相比 AFLNet 有更高的测试吞吐量。

本节实验仍进行 5 轮测试，每轮测试时长限制为 24 h。表 2 展示了每种工具在测试中发现公开漏洞的情况。时间 (Time) 列表示在 5 轮测试中发现漏洞所需的平均时间，以 h、min 和 s 表示，T/O 则表示该工具在测试中未发现相应的漏洞。为评估 IKEChecker 发现漏洞的速度是否明显快于其他工具，使用 A_{V-D} (vargha-delaney, V-D)，统计量进行比较。该统计量反映了从数据组 1 中随机选择的值低于从数据组 2 中随机选择的值的概率。使用 Mann-Whitney U 检验来评估性能增益的统计学显著性^[25]，以 0.95 为标准，

当结果显著时以粗体标记数值。

就检测到的漏洞数量而言，IKEChecker 和 IKEChecker_rand 发现了所有被测试的漏洞，其他测试工具发现的漏洞明显较少。SGFuzz 没有类似消息处理器的功能模块，无法处理加密、解密和认证，导致测试难以进入深层状态，只能发现浅层漏洞。L*+nuXmv 学习算法生成测试序列，但只改变协议消息顺序，没有在更细粒度上执行变异，因此无法检测到像 Libreswan-vul1 (CVE-2023-23009) 这样的漏洞。SECFuzz 缺少有效的反馈引导机制，盲目地生成测试用例，导致在有限的测试时间内发现漏洞数量有限。需要注意的是，AFLNet_IKE 只发现了 4 个新的漏洞中的一个。以漏洞 strongSwan-vul1 为例，触发该漏洞需要将 IKEv1 协议头部的 Exchange Type 字

表2 使用不同工具进行测试所发现的漏洞的统计信息

Table 2 Statistics on vulnerabilities discovered through testing by different tools

漏洞ID	IKEChecker			IKEChecker_rand		AFLNet_IKE		SGFuzz		SECFuzz		L*+nuXmv	
	Time	Time	A_{V-D}	Time	A_{V-D}	Time	A_{V-D}	Time	A_{V-D}	Time	A_{V-D}	Time	A_{V-D}
Libreswan-vul1	6.06h	8.23h	0.60	57.54m	0.08	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00
Libreswan-vul2	59.94m	1.87h	0.88	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00
Libreswan-vul3	6.47h	8.69h	0.60	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00
strongSwan-vul1	19.92m	23.04m	0.76	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00	T/O	1.00
CVE-2023-26463	42.78m	4.19h	0.56	34.92m	0.36	28.12m	0.12	T/O	1.00	T/O	1.00	T/O	1.00
CVE-2022-23094	4.35m	6.47m	0.60	13.41m	0.92	2.32m	0.24	9.23h	1.00	T/O	1.00	T/O	1.00
CVE-2020-1763	31.43s	28.44s	0.32	1.83m	0.64	3.74m	0.80	7.82h	1.00	1.25m	1.00	1.25m	1.00
CVE-2019-12312	23.47s	13.29m	0.72	33.8s	0.60	T/O	1.00	T/O	1.00	1.08m	1.00	1.08m	1.00

段由0x02替换为0x04, 首先需要在长度为几千字节的种子序列中选定该字段, 然后在256个可能取值中选定某一个值, 而且需要顺利通过IKEv1主模式的前两轮交互, 对随机变异条件比较苛刻。这再次表明了本文设计的基于IKE协议语法的变异策略的有效性。

就发现漏洞的时间而言, 基于AFLNet实现的测试工具发现漏洞的效率普遍高于另外两个黑盒测试工具, 主要原因有以下两个: ①在变异中采用堆叠策略, 每次应用多个变异算子; ②采用状态覆盖与代码覆盖相结合的引导机制, 能在更短时间内覆盖更大的搜索空间。SGFuzz相比IKEChecker, 发现几个浅层漏洞所用的时间略少, 这主要是因为SGFuzz采用进程内的基于文件的测试方式, 测试吞吐量远高于其他基于网络通信方式的测试工具。另外, IKEChecker在大多数情况下发现某些漏洞的时间略少于IKEChecker_rand, 这也再次表明基于进化策略的变异调度算法的有效性。一个明显的例外是Libreswan-vul3, 触发此漏洞需要将traffic selector载荷中的长度字段修改为错误值, IKEChecker实现的基于IKE协议语法的变异策略不会对长度字段进行变异, 因此只有Havoc变异才能检测到这种漏洞, 这也是IKEChecker保留Havoc变异策略的重要原因。

总体而言, IKEChecker能够有效发现IKE协议实现中的漏洞, 且在漏洞发现效率方面相较于其他测试工具展现出了一定优势。

5 结束语

本文研究了协议模糊测试领域相关工作, 基于AFLNet模糊测试框架, 开发了一款专用于IKE协议的模糊测试工具IKEChecker, 支持在黑盒和灰盒条件下对IKE协议实现进行测试。针对变异随机性过强问题, 设计了一种基于IKE协议语法的变异策略来提高生成测试用例的有效性和多样性, 并引入基于进化策略的变异调度方案来自动优化变异算子的概率分布, 进一步提高变异效率; 为使模糊测试可以探索深层协议交互行为和状态空间, 设计了消息处理器用以维护协议交互上下文并执行密码学相关的操作。使用IKEChecker对两种支持IKE协议实现的开源软件strongSwan和Libreswan进行了测试, 发现多个未公开漏洞。

当前IKEChecker的整体框架基于AFLNet, 主要考虑通过基于语法的变异提高发现漏洞的效率, 以及通过实现消息处理器使其适用于黑盒场景, 没有对状态表示方式和状态选择算法进行改进。在后续工作中, 考虑实现更细粒度的状态机刻画方式, 从而更全面地检测安全协议实现中的逻辑错误。

参考文献:

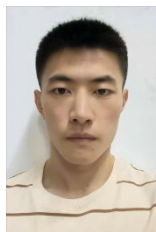
- [1] MANÈS V J M, HAN H, HAN C, et al. The art, science, and engineering of fuzzing: a survey[J]. IEEE Transactions on Software Engineering, 2021, 47(11): 2312-2331.
- [2] PHAM V T, BÖHME M, ROYCHOUDHURY A. AFLNet: a grey-box fuzzer for network protocols[C]//Proceedings of the 2020 IEEE 13th International Conference on Software Testing, Validation and

- Verification. Piscataway: IEEE Press, 2020: 460-465.
- [3] BA J S, MARCEL B, ZAHRA M, et al. Stateful greybox fuzzing[C]// Proceedings of USENIX Security Symposium. Boston: USENIX Association, 2022: 3255-3272.
 - [4] MILLER B P, FREDRIKSEN L, SO B. An empirical study of the reliability of UNIX utilities[J]. Communications of the ACM, 1990, 33(12): 32-44.
 - [5] KAKSONEN R, LAAKSO M, TAKANEN A. System security assessment through specification mutations and fault injection[C]// Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security Issues of the New Century. New York: ACM, 2001: 173-183.
 - [6] LUO Z X, YU J Z, ZUO F L, et al. Bleem: packet sequence oriented fuzzing for protocol implementations[C]// Proceedings of USENIX Security Symposium. Anaheim: USENIX Association, 2023: 4481-4498.
 - [7] NATELLA R. StateAFL: greybox fuzzing for stateful network servers[J]. Empirical Software Engineering, 2022, 27(7): 191.
 - [8] ANDRONIDIS A, CADAR C. SnapFuzz: high-throughput fuzzing of network applications[C]// Proceedings of the 31st ACM Sigsoft International Symposium on Software Testing and Analysis. New York: ACM, 2022: 340-351.
 - [9] LI J Q, LI S Y, SUN G, et al. SNPSFuzzer: a fast greybox fuzzer for stateful network protocols using snapshots[J]. IEEE Transactions on Information Forensics and Security, 2022, 17: 2673-2687.
 - [10] QIN S S, HU F, MA Z Y, et al. NSFuzz: towards efficient and state-aware network service fuzzing[J]. ACM Transactions on Software Engineering and Methodology, 2023, 32(6): 160.
 - [11] LIU D G, PHAM V T, ERNST G, et al. State selection algorithms and their impact on the performance of stateful network protocol fuzzing[C]// Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Piscataway: IEEE Press, 2022: 720-730.
 - [12] LIU D G, ERNST G, MURRAY T, et al. Legion: best-first concolic testing (competition contribution)[C]// Proceedings of International Conference on Fundamental Approaches to Software Engineering. Cham: Springer, 2020: 545-549.
 - [13] HARKINS D, CARREL D. The Internet Key Exchange (IKE)[S]. RFC 2409, 1998: 1-41.
 - [14] KAUFMAN C, HOFFMAN P, NIR Y, et al. Internet Key Exchange Protocol Version 2 (IKEv2)[S]. RFC 7296, 2014: 1-142.
 - [15] MAUGHAN D, SCHERTLER M, SCHNEIDER M, et al. Internet Security Association and Key Management Protocol (ISAKMP)[S]. RFC 2408, 1998: 1-86.
 - [16] WU M Y, JIANG L, XIANG J H, et al. One fuzzing strategy to rule them all[C]// Proceedings of the 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). Piscataway: IEEE Press, 2022: 1634-1645.
 - [17] DE RUITER J, POLL E. Protocol state fuzzing of TLS implementations[C]// Proceedings of USENIX Security Symposium (USENIX Security 15). Washington: USENIX Association, 2015: 193-206.
 - [18] ANGLUIN D. Learning regular sets from queries and counterexamples[J]. Information and Computation, 1987, 75(2): 87-106.
 - [19] JAUERNIG P, JAKOBOVIC D, PICEK S, et al. DARWIN: survival of the fittest fuzzing mutators[J]. arXiv Preprint: arXiv: 2210.11783, 2022.
 - [20] NATELLA R, PHAM V T. ProFuzzBench: a benchmark for stateful protocol fuzzing[C]// Proceedings of the 30th ACM Sigsoft International Symposium on Software Testing and Analysis. New York: ACM, 2021: 662-665.
 - [21] TSANKOV P, DASHTI M T, BASIN D. SECfuzz: Fuzz-testing security protocols[C]// Proceedings of the 2012 7th International Workshop on Automation of Software Test (AST). Piscataway: IEEE Press, 2012: 1-7.
 - [22] TEAN P, JANSSEN R, VAANDRAGER F. Combining model learning and model checking to analyze TCP implementations[C]// Proceedings of International Conference on Computer Aided Verification. Cham: Springer, 2016: 454-471.
 - [23] FITERĂU-BROȘTEAN P, LENAERTS T, POLL E, et al. Model learning and model checking of SSH implementations[C]// Proceedings of the 24th ACM Sigsoft International SPIN Symposium on Model Checking of Software. New York: ACM, 2017: 142-151.
 - [24] GUO J X, GU C X, CHEN X, et al. Model learning and model checking of IPSec implementations for internet of things[J]. IEEE Access, 2019, 7: 171322-171332.
 - [25] KLEES G, RUEF A, COOPER B, et al. Evaluating fuzz testing[C]// Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2018: 2123-2138.

[作者简介]



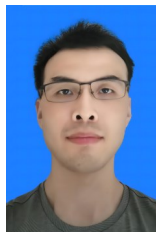
郑永辉 (1976-), 男, 江西乐平人, 信息工程大学副教授、硕士生导师, 主要研究方向为密码学。



赵栋梁 (2000-), 男, 河北保定人, 信息工程大学硕士生, 主要研究方向为网络安全协议。



顾纯祥 (1976-), 男, 安徽霍山人, 信息工程大学教授、博士生导师, 主要研究方向为网络安全、密码学。



张协力 (1992-), 男, 陕西兴平人, 博士, 信息工程大学讲师, 主要研究方向为网络安全协议。