密码学报 (中英文) ISSN 2097-4116 CN 10-1900/TN
Journal of Cryptologic Research, 2024, 11(3): 504–520

E-mail: jcr@cacrnet.org.cn
http://www.jcr.cacrnet.org.cn
Tel/Fax: +86-10-82789618

密码算法库模糊测试技术研究综述*

马福辰, 周远航, 陈元亮, 颜 臻, 姜 宇, 孙家广

清华大学 软件学院, 北京 100084
通信作者: 姜宇, E-mail: jiangyu198964@126.com

摘 要: 密码算法库是提供加密、解密、签名、验证等多种密码学相关功能的基础软件库. 为了保证网络传输的安全性, 很多系统软件都会使用密码算法库来实现数据的隐私性, 以确保数据不被恶意窃取利用. 但是, 密码算法库的实现过程中往往会引入一些漏洞, 这些漏洞会导致系统在使用库中的函数时出现内存崩溃或者加密逻辑失效等后果, 对系统的安全性和可用性带来极大的影响. 模糊测试是一种检测软件实现漏洞的有效技术, 它通过生成大量的测试输入, 观察被测软件的反馈, 进而检测漏洞是否存在. 这种技术已经被应用于密码算法库的漏洞检测中, 并且发现了包括 OpenSSL、SymCrypt、Crypto++ 等常用的密码算法库中的漏洞. 本文通过总结对密码算法库进行高效测试的主要难点, 分析了对密码算法库进行模糊测试的核心需求, 并提出了密码算法库模糊测试工具所面临的主要挑战. 接着, 针对目前常用的 6 款面向密码算法库的模糊测试工具进行了剖析和评估. 最后, 根据目前的工具在漏洞挖掘能力、代码覆盖率和输入有效性等评估指标上的表现, 提出了密码算法库模糊测试未来可能的研究方向和优化策略.

关键词: 模糊测试; 密码算法库; 漏洞挖掘

中图分类号: TP309.7 **文献标识码:** A DOI: 10.13868/j.cnki.jcr.000693

中文引用格式: 马福辰, 周远航, 陈元亮, 颜臻, 姜宇, 孙家广. 密码算法库模糊测试技术研究综述[J]. 密码学报 (中英文), 2024, 11(3): 504–520. [DOI: 10.13868/j.cnki.jcr.000693]

英文引用格式: MA F C, ZHOU Y H, CHEN Y L, YAN Z, JIANG Y, SUN J G. Overview of cryptographic library fuzz testing techniques[J]. Journal of Cryptologic Research, 2024, 11(3): 504–520. [DOI: 10.13868/j.cnki.jcr.000693]

Overview of Cryptographic Library Fuzz Testing Techniques

MA Fu-Chen, ZHOU Yuan-Hang, CHEN Yuan-Liang, YAN Zhen, JIANG Yu, SUN Jia-Guang

School of Software, Tsinghua University, Beijing 100084, China
Corresponding author: JIANG Yu, E-mail: jiangyu198964@126.com

Abstract: The cryptographic algorithm library is a fundamental software library that provides various cryptographic related functions such as encryption, decryption, signature, and verification. In order to ensure the security of network transmission, many system software use cryptographic algorithm libraries to protect data security, to ensure that data is not maliciously stolen or exploited. However, in the implementation process of cryptographic algorithm libraries, vulnerabilities are often introduced, which can lead to memory crashes or encryption logic failures when using functions in the library, greatly affecting the security and availability of the system. Fuzz testing is an effective technique

* 基金项目: 国家重点研发计划 (2022YFB3104000)
Foundation: National Key Research and Development Program of China (2022YFB3104000)
收稿日期: 2024-03-08 定稿日期: 2024-03-29

for detecting software implementation vulnerabilities. It generates a large amount of test inputs, observes the feedback of the tested software, and then detects the vulnerabilities. This technology has been applied in cryptographic algorithm libraries, and many vulnerabilities have been discovered in commonly used cryptographic algorithm libraries such as OpenSSL, SymCrypt, and Crypto++. This paper analyzes the main difficulties in conducting efficient testing on cryptographic algorithm libraries, proposes the requirements for conducting fuzz testing on cryptographic algorithm libraries, and presents the main challenges faced by cryptographic algorithm library fuzz testing tools. This paper also analyzes and evaluates the 6 commonly used fuzz testing tools for cryptographic algorithm libraries. Finally, based on the performance of current tools in evaluating metrics such as vulnerability mining ability, code coverage, and input validity, this paper proposes some possible research directions and optimization strategies for fuzz testing of cryptographic algorithm libraries.

Key words: software testing; cryptographic library; vulnerability detection

1 引言

密码算法库是用于实现各种密码学功能的基础软件库, 一般来说, 一个密码算法库中的函数可以实现加密、解密、签名、验证等各种用于保障数据隐私和安全的功能. 由于密码算法库的需求越来越广泛, 目前实现各种密码学功能的算法也越来越多. 比如用于对数据进行加解密的算法: 分组加密 [1]、序列加密 [2]、公钥加密 [3] 等等. 还有一些算法用于对数据进行验证, 比如 MD5 [4] 等. 为了更好地适配国内自研系统的数据加解密需求, 我国也相继发布了国密算法, 比如 SM2 [5]、SM4 [6] 等. 为了满足不同系统对这些算法的使用需求, 许多密码算法库对各种各样的密码算法进行了实现, 比如 OpenSSL [7]、微软的 SymCrypt [8] 以及 Crypto++ [9] 等.

这些密码算法在理论上都被证明是正确的, 即可以安全有效地完成相应的密码学功能. 然而, 在密码算法库对其进行实现的过程中, 可能会引入很多的代码漏洞导致密码算法无法正确运行. 这些密码算法库中的漏洞往往会造成严重的后果, 让用户遭受损失. 比如在 2014 年 4 月 9 日, OpenSSL 密码库被爆出一个重大漏洞, 即心脏滴血 (Heartbleed) [10]. 该漏洞影响了应用极为广泛的安全传输层协议 TLS 和 DTLS 的心跳过程. 攻击者可以利用这个漏洞获取用户的 id 和密码等隐私数据. 据统计, 这个漏洞影响了全球超过 2/3 的网站, 造成了非常严重的损失. 虽然心脏滴血漏洞引起了人们对密码算法库中漏洞挖掘的重视, 但是时至今日, 仍有许多密码算法库中的严重漏洞被频频爆出. 比如, 2024 年, OpenSSL 中又被爆出重大安全漏洞 CVE-2024-0727 [11], 可以导致使用 OpenSSL 的系统宕机造成 DoS 攻击. 因此, 如何设计高效便捷的密码算法库漏洞检测技术是急需解决的问题.

模糊测试技术是一种动态的软件测试技术. 通过构造大量针对被测系统的输入, 模糊测试工具对被测系统应对这些输入的反馈进行观察分析, 并通过既定的监控策略来判断被测系统是否存在漏洞. 常见的模糊测试工具包括 AFL [12]、LibFuzzer [13]、Peach [14] 等, 模糊测试已经在大量的系统中检测出了许多漏洞. 模糊测试同样被应用于各种密码算法库中, 比如 CryptoFuzz [15] 在常用的密码库中检测出了数十个之前未知的漏洞. 此外, 学术界的工具 CLFuzz [16] 通过语义理解的方式, 生成大量的高效测试输入, 并可以通过交叉验证的方式, 对模糊测试过程中产生的输入进行密码逻辑的执行正确性检查.

然而, 目前的模糊测试工具仍然面临着一些挑战, 限制其在密码算法库上的测试效果. 本文为了研究模糊测试技术在密码算法库上应用过程中存在的难点以及改进的方向, 分析总结了 6 个常用的面向密码算法库的模糊测试工具, 通过对它们进行分类的方式, 探究其采用的技术路线, 并设计实验来评估它们目前的测试效果. 本文在模糊测试工具的漏洞发现能力、代码覆盖率以及生成输入的有效性等维度上对这些工具进行了评估, 并给出了密码算法库模糊测试的未来研究方向.

本文的主要贡献包括如下三个方面:

(1) 总结了模糊测试技术应用于密码算法库测试的需求, 并提出流程中不同步骤的关键技术难点.

(2) 对 6 个常用的密码算法库的模糊测试工具进行了研究, 分析了这些工具所采用的核心技术、取得的效果以及可能的不足.

(3) 通过实验评估了这些工具在密码算法库测试上的效果, 从漏洞发现数量、代码覆盖率以及生成的测试输入的合法性等方面进行了分析, 并给出密码算法库模糊测试的未来发展方向和建议.

2 密码算法库模糊测试技术概述

2.1 模糊测试技术

模糊测试技术是一种应用于计算机软件测试的常用手段. 从测试输入生成的角度进行划分, 模糊测试技术主要分为基于生成的技术和基于变异的技术. 基于生成的模糊测试技术基于测试输入的格式规范, 生成大量符合要求的测试输入. 这种测试方法往往用于对测试输入要求高的软件测试中, 比如通讯协议测试. 而基于变异的模糊测试技术则通过对已有测试输入的变异, 来产生大量的新的测试输入. 为了产生高质量的新输入, 基于变异的模糊测试技术一般都会采用一些反馈机制来指导变异过程. 最为常见的反馈机制就是利用被测软件的覆盖率信息来进行反馈. 图 1 展示了基于覆盖率反馈的变异模糊测试的流程.

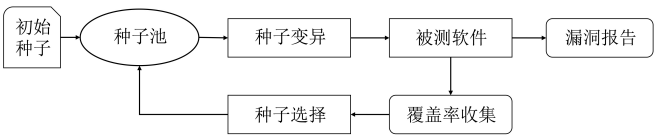


图 1 基于变异的模糊测试的一般流程

Figure 1 Overall workflow of mutation-based fuzzing

如图 1 所示, 首先测试人员需要基于被测软件构造一些初始种子, 并将这些初始种子放入种子池中. 在每一轮的模糊测试过程中, 模糊测试器从种子池中选出若干种子进行变异, 这里的变异指的是对种子的内容进行修改的过程. 常见的种子变异方式有字节翻转以及位翻转等. 经过变异后的种子会被输入给被测软件进行执行, 在执行过程中, 模糊测试器通过一些检测器判断当前的软件是否存在漏洞, 如果存在漏洞则形成报告. 如果不存在漏洞, 模糊测试器会通过预先插入的桩点对被测软件的执行覆盖率进行收集. 基于这个覆盖率, 模糊测试器会对这一轮的种子进行选择, 判断它们是否是有价值的种子. 如果是有价值的种子, 则将它们存放入种子池中用于下一轮的变异测试, 否则则丢弃这些种子. 种子选择的常见指标是当前种子是否使被测软件的执行覆盖率得到提升, 如果种子执行到了被测软件之前未被覆盖的代码逻辑, 则认为这是一个好的种子.

2.2 密码算法库模糊测试的需求和挑战

2.2.1 需求

密码算法库作为许多系统的基础组件, 其安全性相关的测试需求主要体现在如下几个方面.

第一个方面是内存安全. 由于密码算法库实现的语言多种多样, 包括 C++、C 语言等. 这些语言实现的代码中, 由于对内存访问控制没有严格的要求和限制, 例如数组越界、内存泄漏、内存释放后使用等漏洞时常出现. 密码算法库的实现过程中, 如果隐含了这些内存相关的漏洞, 往往会导致密码算法的运行崩溃, 造成使用密码算法库的系统宕机. 更为严重的是, 这些内存漏洞有些还会造成远程代码执行 (RCE) 攻击, 使攻击者利用恶意构造的内存片段进行恶意代码执行, 对系统造成更大的影响和危害. 著名的 OpenSSL 中的心脏滴血漏洞 [10] 就属于内存漏洞的一种. 在心脏滴血漏洞中, 由于没有对心跳包中长度字段的内容进行有效的边界检查, 攻击者可以访问到受害用户的私有内存信息, 导致信息泄露.

第二个方面是密码逻辑安全. 密码算法库所包含的密码算法多种多样, 有加密算法、签名算法等. 这些算法在理论层面都被证明是正确的, 但是在实现层面, 由于代码实现的易错性, 很容易出现加密逻辑与理论不符的情况. 这种实现上的漏洞, 会导致密码算法库在执行过程中无法按照既定密码逻辑输出正确的结果. 这种类型的漏洞会对应用密码算法库的系统造成严重的危害, 系统会由于加密或者解密失败, 导致信息泄露或者处理异常, 更为严重的是, 如果某些漏洞使系统的验证功能出现错误, 将某些非法的签名验证为合法的, 就会使得系统的安全性受到极大的挑战. 密码算法库逻辑安全漏洞的出现也很频繁, 比如 2023 年的一个 OpenSSL 中的漏洞 CVE-2023-5363 [17], 由于在处理密钥和初始化向量 (IV) 长度时出现

错误, 导致某些对称密码在初始化期间存在一些潜在的截断. 这种截断可能导致密钥生成过程中的非唯一性, 从而导致某些密码算法的输出结果异常.

第三个方面是密码算法库兼容性安全. 由于密码算法库的种类繁多, 实现语言也多种多样, 而且版本迭代非常快. 这导致在一个大型的分布式系统中, 可能存在不同节点采用的密码算法库不同的情况. 此外, 在协议通讯的过程中, 通讯的客户端和服务端所使用的密码算法库实现也可能完全不同. 这就要求, 不同的密码算法库在对同一种密码算法的实现上保持兼容性. 比如对称加密算法中, 用 OpenSSL 中的 AES 计算出来的密文, 需要与 SymCrypt 中的 AES 用同样的明文和密钥计算出来的密文相同. 用 OpenSSL 3.1 版本的 AES 计算出来的密文, 也需要与 3.0 版本的 AES 计算的结果相同. 这种兼容性对于系统的稳定运行是至关重要的.

2.2.2 挑战

为了对密码算法库进行高效的模糊测试, 有一些需要解决的挑战, 具体主要包含如下几个方面.

第一个方面的挑战是测试驱动的构建. 为了对密码算法库的各种算法进行模糊测试, 一个最为重要的工作是编写这些算法的测试驱动. 测试驱动是一个测试入口程序, 这个程序会调用对应密码算法的各种函数和功能, 并将它们组织起来用于提高该算法的测试效果. 一般来说, 一个针对密码算法库的测试驱动包含如下几个流程: (1) 密码语料准备. 即对驱动所要测试的密码算法所需的语料进行初始化, 比如种子构造、椭圆曲线初始化等等. (2) 测试算法执行. 即对被测的密码算法函数进行调用, 参数中传入在上个步骤中构造的初始预料, 以及测试引擎构造的变异后的测试输入. (3) 资源释放. 即对测试算法所使用的资源进行释放, 避免多次调用驱动函数造成资源的浪费. 图 2 中所示的代码片段展示了模糊测试工具 CryptoFuzz 在 OpenSSL 上对 Digest 操作编写的测试驱动. 如第 3 行所示, 这个驱动首先获取当前传入的操作的数据信息. 之后, 代码的第 7–12 行基于这个信息开始 Digest 操作的初始化过程, 这属于测试驱动的密码语料准备阶段. 这个过程会对数据内容进行检查. 然后在第 13–16 行, 算法会进行 Digest 的处理, 利用函数 'EVP_DigestFinal_ex' 完成密码算法的逻辑, 这属于测试驱动的算法执行阶段. 最后, 在 17–23 行, 驱动执行该算法的退出操作, 并返回相关结果, 这属于测试驱动的资源释放阶段.

```
1  std::optional<component::Digest> OpenSSL::OpDigest(operation::Digest& op) {
2      std::optional<component::Digest> ret = std::nullopt;
3      Datasource ds(op.modifier.GetPtr(), op.modifier.GetSize());
4      util::Multipart parts;
5      CF_EVP_MD_CTX ctx(ds);
6      const EVP_MD* md = nullptr;
7      /* 初始化 */
8      {
9          parts = util::ToParts(ds, op.cleartext);
10         CF_CHECK_NE(md = toEVPMD(op.digestType), nullptr);
11         CF_CHECK_EQ(EVP_DigestInit_ex(ctx.GetPtr(), md, nullptr), 1);
12     }
13     /* 算法处理 */
14     for (const auto& part : parts) {
15         CF_CHECK_EQ(EVP_DigestUpdate(ctx.GetPtr(), part.first, part.second), 1);
16     }
17     /* 算法结束 */
18     {
19         unsigned int len = -1;
20         unsigned char md[EVP_MAX_MD_SIZE];
21         CF_CHECK_EQ(EVP_DigestFinal_ex(ctx.GetPtr(), md, &len), 1);
22         ret = component::Digest(md, len);
23     }
24 end:
25     return ret;
26 }
```

图 2 CryptoFuzz 对 OpenSSL 中 Digest 操作编写的测试驱动

Figure 2 Fuzzing driver for operation digest in OpenSSL from CryptoFuzz

总的来说, 一个测试驱动模拟了现实应用调用密码算法库中某些算法的逻辑和过程. 这样的驱动可以

保证测试的执行逻辑跟实际使用逻辑的一致性, 确保测出的漏洞是真实的. 但是, 由于密码算法库中算法繁多, 不同算法的驱动程序不尽相同. 为了高效对密码算法库展开模糊测试, 需要利用领域知识, 对每个密码算法编写驱动, 并尽可能多地调用这个算法的不同使用场景, 来保证测试的高效性. 这往往需要耗费大量的人工成本, 所以对于密码算法库的测试来说, 构造高质量的测试驱动是一个严峻的挑战.

第二个方面的挑战是测试输入的合法性. 为了高效触发密码算法库的实现漏洞, 需要构造大量合法的测试输入来覆盖密码算法实现的深层逻辑. 由于密码算法的输入参数类型多样、语法规则丰富, 通常具有严格的格式检查和语法要求. 如果输入的参数不能满足密码算法的参数检查, 那么就无法触发密码算法的执行逻辑, 导致测试效率低下. 图 3 中的代码片段展示了 OpenSSL 中对不同 cipher 模式下 key 的长度的检查过程. 如第 5 行和第 6 行所示, OpenSSL 会根据传入的 cipher 模式 c, 来用函数 'EVP_CIPHER_CTX_get_key_length' 获取该模式应该传入的 key 的长度. 这个长度会与外部传入的 key 的实际长度进行比对, 如果这两个长度值相同, 那么 OpenSSL 就认为当前传入的 key 的长度是合法的, 并返回 1. 在第 15–16 行可以看到, 如果这个检查没有通过, 那么 OpenSSL 将会处理异常, 而不会进行后续的加密逻辑. 因此, 对密码算法库的模糊测试来说, 生成符合其参数规范的测试输入是很重要的.

```
1  int EVP_CIPHER_CTX_set_key_length(EVP_CIPHER *c, int key1) {
2      if (c->cipher->prov != NULL){
3          ...
4          // 检查当前传入的cipher模式的key的长度是否符合要求
5          if (EVP_CIPHER_CTX_get_key_length(c) == key1)
6              return 1;
7          ...
8      }
9      ...
10     ERR_raise(ERR_LIB_EVP, EVP_R_INVALID_KEY_LENGTH);
11     return 0;
12 }
13
14 // 对于EVP_aes_256_cbc()来说, key的长度应该是256位, 即32个字节
15 if(1 != EVP_CIPHER_CTX_set_key_length(EVP_aes_256_cbc(), key_length))
16     handleErrors();
```

图 3 OpenSSL 中对不同 cipher 模式下的 key 的长度的检查

Figure 3 Key length verification of OpenSSL for different cipher modes

但是, 由于密码算法种类繁多, 每种算法对输入参数的长度、格式等要求不尽相同. 而且与通信协议、操作系统内核中的系统调用等包含标准输入格式规范的软件不同, 密码算法库的输入规范往往没有明确的输入规范说明. 只能通过相关代码中检查函数的信息来进行格式规范提取并对测试输入的生成进行指导. 如何生成高质量的合法的密码算法库测试输入是对其进行高效模糊测试的重大挑战.

第三个方面的挑战是测试输入的高效性. 为了触发深层次的加密处理逻辑, 在合法的测试输入的基础上, 如何变异出高效测试输入是尤为关键的. 传统的模糊测试工具 AFL 和 LibFuzzer 等, 往往采用字节翻转或者位翻转的方式来完成输入的变异. 这种变异方式可以快速产生大量新的测试输入, 让密码算法库进行执行, 并快速提高对密码算法库的测试覆盖率. 但是, 很多情况下, 密码算法库的漏洞只能在一些特殊的测试输入下被触发. 比如图 4 中的例子, 展示了一个 OpenSSL 中正数溢出漏洞的代码片段. 代码中的函数 'evp_EncryptDecryptUpdate' 根据输入的参数 inl 计算一个返回值 outl 用于接下来的密钥计算过程. 当传入的 inl 很大, 接近正数最大值的时候, 代码中的第 26 行, 可能会在加法运算的时候出现溢出, 导致存入指针 outl 的值为负数, 影响后续的计算过程. 为了解决这个问题, 开发人员在第 13–16 行添加了修复的代码, 即在计算 outl 存储的值之前, 先判断当前的输入是否会造成溢出, 如果会溢出, 那么抛出一个异常, 而不会返回正常值.

图中所示的漏洞在检测过程中并不容易被触发, 这是因为触发它需要对输入的参数进行特定性的变异. 比如说这个例子中, 就需要在对 inl 进行变异时, 尝试变成极大值, 才能触发这个漏洞. 但是目前的模糊测试工具比如 AFL 和 LibFuzzer 的变异策略很难变异出这种特定的数值. 在密码算法库各种函数的参数中, 这种需要特定取值才能触发的漏洞很多, 这就需要模糊测试工具在进行种子变异的时候去考虑如何

将随机的去值变异和特殊值变异相结合.

```
1   static int evp_EncryptDecryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int
        *outl, const unsigned char *in, int inl)
2   {
3     ...
4     if (i != 0) {
5       if (bl - i > inl) {
6         memcpy(&(ctx->buf[i]), in, inl);
7         ctx->buf_len += inl;
8         *outl = 0;
9         return 1;
10      } else {
11        j = bl - i;
12        // 必须保证剩下的块长度:(inl - j) & ~(bl - 1)
              加上ctx->buf处理的块长bl不超过INT_MAX
13        +          if (((inl - j) & ~(bl - 1)) > INT_MAX - bl) {
14          +              ERR_raise(ERR_LIB_EVP, EVP_R_OUTPUT_WOULD_OVERFLOW);
15          +              return 0;
16          +          }
17        memcpy(&(ctx->buf[i]), in, j);
18        ...
19      }
20    }
21    ...
22    if (inl > 0) {
23      if (!ctx->cipher->do_cipher(ctx, out, in, inl))
24      return 0;
25      // inl如果很大, *outl可能会超过正整数最大值, 变成负数
26      *outl += inl;
27    }
28  }
```

图 4 OpenSSL 中的一个整数溢出漏洞及其修复

Figure 4 An integer overflow bug in OpenSSL with patch

第四个方面的挑战是检测漏洞类型的多样性. 对于密码算法库来说, 内存安全性、实现正确性和兼容性都是非常重要的. 对内存安全来说, 可以对密码算法库进行 ASAN [18] 等检测器的插桩, 借助生成的高质量测试输入来完成对内存安全的覆盖和检测. 但是对于密码算法库的实现正确性来说, 相关的检测器很难构造. 这往往需要根据目标密码算法的功能来进行特定的构造. 比如对于对称加密算法来说, 需要检查生成的密文能否被正确解密. 对于非对称加密算法来说, 需要检查签名的内容能否被正确验证. 这种逻辑上的检查往往需要领域知识和手动的定义, 但是逻辑漏洞对密码算法库上层系统是至关重要的, 往往会造成严重的损失.

比如 2023 年, 在 OpenSSL 中的一个漏洞会使得计算出的密钥机密性丧失, 这个漏洞被分配了一个 CVE 编号 CVE-2023-5363 [17]. 这个漏洞出现的原因如图 5, 在第 3–7 行, 函数提供的参数数组在密钥和 IV 建立后进行处理, 这就导致参数数组中传入的长度等属性没有作用到密钥和 IV 上, 导致生成的密钥出现截断. 对这种类型的漏洞的检测是比较困难的, 因为这需要检测当前生成的密钥是不是存在截断或者额外的字节, 这些结果可能最终会反映在加密失效或者密钥相似等现象上. 一个模糊测试检测器需要对这些现象进行有效的识别和分析, 同时要尽量避免误报和漏报的出现, 这对模糊测试的有效性至关重要, 同时也是一个很大的挑战.

2.3 密码算法库模糊测试典型工具

近年来工业界和学术界都涌现了一些典型的面向密码算法库的模糊测试工具, 它们在主流的密码算法库上发现了很多严重漏洞. 本文对这些工具在上述四个主要挑战上的解决方案进行了分析整理, 表 1 展示了这些工具的基本情况.

表中介绍了 6 款面向密码算法库的模糊测试工具, 并从驱动构造、测试输入合法性、测试输入有效性和漏洞检测这四个核心挑战上介绍这些工具的解决思路.

```
    return ctx->cipher->einit(ctx->algctx,
    key,
    key == NULL ? 0
    : EVP_CIPHER_CTX_get_key_length(ctx),
    iv,
    iv == NULL ? 0
    : EVP_CIPHER_CTX_get_iv_length(ctx),
    params);
```

图 5 OpenSSL 中的一个漏洞使得生成密钥的机密性丧失

Figure 5 A vulnerability in OpenSSL which can leads to an instant loss of confidentiality

表 **1** 密码算法库模糊测试典型工具汇总

Table 1 Typical fuzzing tools for cryptographic libraries

工具名称	年份	通用性		驱动构造		测试输入合法性		测试输入有效性		漏洞检测	
		支持算法库数量	适配新算法库	驱动数量	编写新驱动方法	合法性支持	合法性输入生成	有效性支持	有效性输入生成	支持漏洞类型	检测方式
CryptoFuzz [15]	2019	107	实例化模板类	896	算法调用场景构建	部分支持	字节流划分	不支持	不支持	内存漏洞 逻辑漏洞 兼容性漏洞	ASAN 内部验证 差分测试
CLFuzz [16]	2023	54	实例化模板类	530	算法调用场景构建	支持	规范信息提取	支持	极大值 极小值 特殊值	内存漏洞 逻辑漏洞 兼容性漏洞	ASAN 交叉验证 差分测试
CDF [19]	2017	9	遍历调用密码算法	9	遍历调用密码算法	支持	手动配置	部分支持	手动配置	逻辑漏洞 兼容性漏洞	断言 差分测试
WycheProof [20]	2016	8	用例编写	32	用例编写	不支持	手动配置	支持	手动配置	逻辑漏洞	断言
Jin et.al [21]	2022	/	实例化模板类	/	算法调用场景构建	部分支持	字节流划分	支持	符号执行	内存漏洞 逻辑漏洞 兼容性漏洞	ASAN 内部验证 差分测试
ecfuzzer [22]	2018	11	实例化模板类	11	算法调用场景构建	支持	固定格式	不支持	不支持	内存漏洞 逻辑漏洞	ASAN 椭圆曲线点坐标检查

CryptoFuzz 是一款开源工具, 于 2019 年发布. 迄今为止, CryptoFuzz 在包括 OpenSSL [7]、LibreSSL [23]、BoringSSL [24] 等常用的密码算法库中发现了 193 个漏洞. 从通用性的角度来看, CryptoFuzz 目前支持对 107 个密码算法库的测试, 通过定义一个模版类 'Module', 来实现高扩展性. 当适配一个新的密码算法库的时候, CryptoFuzz 只需要对这个模板类进行实例化即可, 适配难度相对较低. 从驱动构造的角度, CryptoFuzz 利用领域知识, 对不同的密码库中所包含的算法进行人为的驱动构造. 目前, CryptoFuzz 已经编写了 896 个针对不同密码库的测试驱动, 并对这些密码库中的不同算法进行了调用和串联. 如果要构造一个新的驱动, 只需要在对应密码库的驱动中, 完成对算法对应函数的调用即可, 并实现一些这个算法的使用场景, 就可以快速接入 CryptoFuzz 的模糊测试引擎. CryptoFuzz 是利用 LibFuzzer [13] 来完成模糊测试的变异等任务的, 但是为了维持变异输入的合法性, CryptoFuzz 会根据被测算法的函数参数进行字节流的字段划分, 保证 LibFuzzer 产生的变异输入, 可以被正确地灌入进被测函数中, 完成被测函数的调用. 但是, CryptoFuzz 并没有考虑被测函数中对参数的检查条件, 比如 key 和 iv 的长度限制等等, 所以它生成的测试输入并不总是合法的. 此外, CryptoFuzz 没有考虑输入的有效性, 它完全依靠 LibFuzzer 的变异策略, 而不会考虑一些输入的特殊取值. 目前, CryptoFuzz 支持内存相关的漏洞的检测, 比如节点崩溃、卡住等. 此外, 它还支持内部逻辑检查, 比如使用对称加密算法的加密, 之后使用该算法的解密, 看是否可以完成正常解密. 最后, 它还通过差分测试的方法支持对兼容性漏洞的检测.

CLFuzz 是 2023 年发表在 TOSEM 期刊上的一个工具, 它在 CryptoFuzz 的基础之上完成. CLFuzz 目前支持 54 个密码算法库, 与 CryptoFuzz 类似, 如果要适配一个新的密码算法库, CLFuzz 也可以通过实例化 Module 这个模版类的方式完成. 在驱动构造方面, CLFuzz 采用跟 CryptoFuzz 一样的方法来手动构建驱动. 与 CryptoFuzz 相比, CLFuzz 的主要改进点在于对测试输入合法性、测试输入有效性和漏洞检测类型这三方面上. 为了更好地满足测试输入的合法性, CLFuzz 对密码算法库中输入参数的规范信息进行了抽取, 在生成每一轮的测试输入的时候, 会结合抽取的规范来完成. 这意味着 CLFuzz 生成的测

试输入大部分是可以通过诸如长度等规范检查的. 这就极大地提高了测试输入的合法性, 加快了测试效率. 此外, 为了产生更为有效的测试输入, CLFuzz 在生成测试输入的时候, 采用极大值、极小值、空值等特殊内容来对输入参数进行填充, 这样的方法可以触发一些极端情况下的漏洞. 在漏洞检测方面, CLFuzz 除了支持 CryptoFuzz 的漏洞之外, 还设计了一种交叉验证的方法, 比如对于对称加密算法来说, 用某个算法的加密功能来生成密文, 用该算法的另一个不同的实现来完成解密, 检查是否可以解密密文. CLFuzz 已经在各个常用的密码库上发现了 12 个之前未知的密码算法库漏洞.

CDF 是一款 2017 年发布的开源测试工具. CDF 目前支持 9 个密码算法库, 如果要进行新的密码算法库适配, CDF 需要遍历该算法库的每个密码算法, 并在对应的驱动中添加对该算法的调用完成适配, 适配难度相对较大. CDF 的驱动是按照密码算法编写的, 对于某个密码算法来说, CDF 会选择所有实现了这个算法的密码库, 并在驱动中对这些密码库按照既定的驱动规则进行调用. 对 CDF 来说, 如果要编写一个新的驱动, 需要首先设计一个密码算法的调用逻辑, 然后按照这个逻辑将所有密码库中对应的算法功能进行调用. CDF 支持测试人员手动配置输入的参数, 因此, 在测试输入合法性方面可以得到保障. 对于测试输入的有效性来说, CDF 的配置可能会覆盖到一些特殊的取值, 因此在测试的时候, 可能可以对某些特殊情况的漏洞进行覆盖, 因此 CDF 的部分测试输入有效性很高. 与前两个工具不同, CDF 目前支持的漏洞类型主要是逻辑漏洞, 主要是靠驱动中的断言来实现的. 测试人员在驱动中需要人为定义行为正确与否, 完成对相应密码算法的断言.

WycheProof 是谷歌公司的开发者设计的一款面向密码算法库的测试工具, 于 2016 年开源发布. 它目前支持 8 种密码算法库的测试, 如果要适配一个新的密码算法库, 需要从头开始编写关于该库算法的测试用例, 并实现对该算法的调用和断言. WycheProof 为各种密码算法的实现编写了大量的测试用例, 这些用例可以确保测试输入的合法性和有效性. WycheProof 支持的漏洞类型也集中在逻辑类型的漏洞上, 它主要利用之前的一些攻击事件来设计驱动逻辑和断言, 通过断言来检查当前的密码库实现是否存在与之前被攻击者利用的相同的漏洞.

Jin 等研究者在 2022 年发表的论文中提出了一种混合模糊测试的方法用于密码算法库的测试. 这种方法也是基于 CryptoFuzz 改进的. 这个方法的核心思想是利用混合符号执行与模糊测试相互结合的办法来提高测试的覆盖率和漏洞发现能力. 该方法目前没有开源, 因此无法获取其目前支持的密码库的数量. 它适配到一个新的密码算法库上的难度跟 CryptoFuzz 类似. 此外, 在驱动构造、测试输入合法性和漏洞检测这几个维度上的支持程度跟 CryptoFuzz 基本一致. 但是在测试输入有效性上, 该方法通过混合符号执行约束求解, 可以提供给模糊测试一些高效的测试输入, 进而覆盖更多的密码算法代码.

ecfuzzer 是 2018 年开源发布的一款模糊测试工具. 它面向椭圆曲线相关的密码算法库进行测试. 目前, 它支持对 11 种密码库的测试, 与 CryptoFuzz 类似, ecfuzzer 也支持使用模版类来对新的密码算法库进行拓展. 由于只面向椭圆曲线算法, ecfuzzer 的测试驱动都是手动构造且满足如下的流程: (1) 驱动加载椭圆曲线模型; (2) 驱动调用密码算法实现计算两个大数的标量乘法运算; (3) 驱动检查计算的返回值和结果, 判断是否存在漏洞. 因此, 编写一个新的驱动的方法也相对比较简单, 只需要按照这个步骤完成对目标算法实现的调用和检查即可. ecfuzzer 规定, 传入的参数前两个字节表示当前的椭圆曲线模型, 后面两个字节分别代表用于计算的两个大数, 因此它产生的输入合法性很强. 但是由于它也是利用 LibFuzzer 直接对测试输入进行变异的, 其生成测试输入的有效性不高. 对漏洞检测方面, 目前 ecfuzzer 支持对内存漏洞、兼容性漏洞和逻辑漏洞的检测.

3 密码算法库模糊测试的关键技术

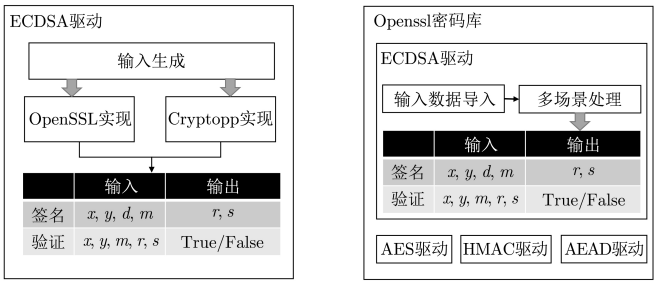
为了应对密码算法库模糊测试的关键挑战, 目前的典型工具提出了一些关键的技术方案. 本节将按照第 2 节提出的四个挑战来分别对目前典型工具所采用的技术进行详细描述.

3.1 测试驱动构建技术

对于密码算法库模糊测试来说, 测试驱动是最为关键的测试入口, 驱动质量的优劣直接影响模糊测试的最终效果. 因此, 几乎所有的测试工具都对测试驱动进行了构建和定义. 一般来说, 一个好的针对密码库的测试驱动需要包括如下几个特征: (1) 场景多样: 驱动中对密码算法库的调用场景多样, 对密码算法在实

际运用中的各个场景都有所涉及; (2) 断言丰富: 一般的逻辑漏洞都需要驱动中的断言来进行检查, 为了能检测更多的逻辑漏洞, 驱动中需要结合密码算法的具体语义, 设计尽可能丰富的断言; (3) 资源管理: 驱动程序在测试过程中会被频繁启动, 驱动中会申请和使用很多内存资源等等, 需要在驱动程序退出之前对他们进行释放. 需要尽量避免驱动程序自己存在一些漏洞影响对密码算法库的测试进程.

目前的典型工具都包含大量的测试驱动, 从设计的结构上来划分, 目前工具中的驱动组织形式可以分为两大类, 如图 6 所示. 第一类是: 针对密码算法的多实现驱动组织形式. 采用这种类型的典型工具主要是 CDF. 在这种组织形式下, 一个密码算法的驱动会包含若干个密码库对它的具体实现的调用. 以 ECDSA 签名算法为例, 如图 6 (a) 所示, ECDSA 驱动中首先构造对当前密码库调用的输入, 在 CDF 中, 这往往是从测试人员配置的文件读入的. 之后, 驱动将这些输入灌输到不同密码库的实现中去, 比如调用 OpenSSL 对 ECDSA 的实现, 再调用 Crypto++ 对 ECDSA 的实现等. 最后, 通过比较这些实现对相同输入的输出结果, 来判断是否存在漏洞. 这种组织形式的驱动, 可以包含丰富的场景, 但是在断言的层面, 仅仅适用于差分比较, 对逻辑断言的支持程度相对较弱.



(a) 密码算法的多实现驱动组织形式 (b) 密码库的多算法驱动组织形式

图 6 两种模式的驱动组织形式

Figure 6 Two types of cryptographic library drivers

另一种驱动的组织形式是: 针对密码库的多算法驱动组织形式, 如图 6 (b) 所示. 这种组织形式常见于 CryptoFuzz 及其拓展的相关工具, 比如 CLFuzz、Jin 等的工具中. 此外, WycheProof 和 ecfuzzer 也采用这种方式来组织驱动架构. 具体来说, 这种组织形式对一个密码库中所有的密码算法进行驱动的设计. 以 ECDSA 驱动为例, 首先, 通过一个字节流导入驱动的输入数据, 之后对该算法的多场景进行处理. 对于 ECDSA 来说, 可能包含签名 (sign)、随机数签名 (sign_with_nonce) 等等. 最后, 根据 ECDSA 的签名逻辑, 对所有场景的输出结果进行验证, 比如验证签名的输出是否是空、输出是否可以被正确验证等等. 这种形式的驱动场景丰富, 而且断言构造能力强, 但是资源管理方面相对更为复杂, 因为场景多样, 涉及到的资源也很多, 需要在退出阶段对这些资源进行依次的释放.

对于驱动构建技术来说, 目前的所有典型工具都是采用手动构造的方式来完成的. 虽然手动构造的驱动准确性和丰富程度高, 但是构造难度大, 需要测试人员对被测密码算法库有很深的理解. 目前有一些工作, 可以对库代码进行自动的模糊测试驱动生成. 比如 Daisy [25] 可以动态抽取对象使用序列来自动化构建测试驱动. 但是这些工作在对密码算法库进行驱动构造的时候, 会面临一些问题. 首先它们无法有效理解密码库的语义, 这会影响驱动调用的场景和生成的断言. 此外, 自动化工具生成的驱动正确性难以保证, 可能有些驱动自身存在漏洞, 导致测试无法进入到库代码本身. 如何有效解决这些问题, 并对密码算法库的驱动进行自动化、高效的构造, 避免人工成本的同时保障驱动的高质量, 是未来密码库驱动构造技术的主要挑战和发展方向.

3.2 测试输入合法性保障技术

测试输入的合法性对密码库的模糊测试也是至关重要的. 对测试输入的合法性的考量主要集中在两方面: (1) 首先是输入参数的类型限制. 如果输入的参数类型不正确, 那么驱动在调用密码算法实现的时候会失败. (2) 其次是输入参数的格式校验. 如果参数的格式不满足要求, 则会在密码算法库的浅层代码逻辑中抛出异常, 无法深入到后续的加密逻辑中. 目前的典型工具都对测试输入的合法性进行了一些保障. 比如 CryptoFuzz, 它对 LibFuzzer 产生的字节流进行了字段分割, 来满足输入参数的类型限制. 但是

CryptoFuzz 没有对输入参数的格式校验进行满足. 与 CryptoFuzz 类似, Jin 等人的输入合法性也是仅仅考虑了参数的类型而没有考虑参数的格式要求.

CDF 则通过测试人员手动配置的配置文件, 来生成符合参数类型和格式规范要求的测试输入. 与之类似地, WycheProof 也是通过手动构造每一个测试用例的输入来满足输入的合法性的. 这种办法虽然可以满足测试输入的合法性要求, 但是需要大量的人工成本和工作量. 对于新适配的密码算法库, 需要花费大量的精力来构造配置或者测试输入, 满足对合法性的需求. ecfuzzer 由于只面向椭圆曲线算法, 所以它的输入格式相对比较统一. 因此 ecfuzzer 规定输入的参数前两个字节是椭圆曲线的模型, 第三个字节是输入的第一个大数, 最后一个字节是输入的第二个大数. 因此输入的合法性是可以完全满足的.

CLFuzz 则采用一种语义自动抽取的方法来满足输入的合法性. 图 7 展示了这种方法的技术路线. CLFuzz 首先会识别被测密码算法函数的函数签名, 之后根据函数签名中的参数类型, 来生成符合类型的参数输入. 此外, 为了对输入参数的格式规范进行识别和生成, CLFuzz 识别与语法规范相关的函数比如 'EVP_CIPHER_get_key_length' 和 'EVP_CIPHER_get_iv_length' 等, 并在开始进行模糊测试之前, 对这些函数进行调用, 获取不同字段的长度等语法规范信息. 通过这种方法, CLFuzz 可以自动化地抽取函数输入参数的语法规范, 并满足输入合法性的需求. 这种方法对于新适配的密码库来说也可以很方便地被应用, 从而提取输入规范的要求.

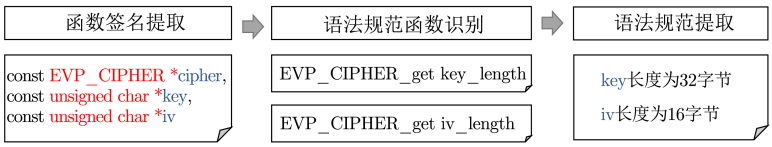


图 7 CLFuzz 所采用的输入参数规范自动抽取技术

Figure 7 Input specification auto-extraction technique of CLFuzz

对于测试输入合法性保障技术来说, 目前的典型工具通过手动配置、自动抽取等技术来提高输入的合法性. 但是, 目前的手动配置技术成本较高, 适配难度较大, 而自动抽取技术也面临着不够完备、精准度不够等问题. 第 4 节将对目前技术输入种子的有效性进行实验评估, 并分析目前这些技术的效果. 未来对密码算法库的输入合法性保障上需要进行更为准确、高效的自动化抽取方法, 比如利用静态分析的技术对合法性检查进行扫描, 再借助大语言模型等技术进行规则的抽取等, 来保障生成的种子一定是合法的.

3.3 测试输入有效性保障技术

在保障测试输入的合法性之后, 测试输入的有效性对模糊测试也是非常重要的. 有些漏洞需要在特定的输入下才能被触发, 如何构造这样的输入对密码库的测试是很困难的问题. 目前的工具对测试输入的有效性保障还不是很全面. CryptoFuzz 和 ecfuzzer 不支持对测试输入有效性的考量. 为了补足 CryptoFuzz 的这个问题, CLFuzz 利用特殊构造的一些数据值来对输入的有效性进行支持. 在 CLFuzz 中, 测试输入的取值可能是如下几种类型: (1) 极大值: 数值类型的极大值, 比如 INT_MAX 等; (2) 极小值: 数值类型的最小值, 比如 0; (3) 特殊值: 一些特殊字段的特殊取值, 比如空值、某些标志位字段的 0xff 等等. 通过这些取值的约束, CLFuzz 可以对密码算法的边界情况进行探索, 满足输入有效性的要求. CDF 则利用手动配置的输入来部分支持有效性. 使用 CDF 的测试人员需要在配置输入范围的时候, 考虑一些可能存在的边界条件或者特殊值. 但是这种支持可能无法覆盖很多特殊的取值, 所以输入的有效性也不能很好保障. WycheProof 也采用手动配置的方法, 但是由于 WycheProof 对每个密码算法都包含大量的测试样例, 这些样例中的输入有很多特殊值和边界值. 这些输入大部分是由之前真正攻击的场景中提炼出来的, 有效性很高.

Jin 等研究人员开发的工具利用混合符号执行来生成更加有效的输入. 这种方式的具体做法是通过符号执行来收集约束并求解, 并对模糊测试的输入生成进行辅助, 方便测试引擎快速进入某些难以进入的分支. 该方法采用了 Intriguer [26] 作为混合符号执行的引擎. 该引擎利用如图 8 的方式来对路径约束进行求解并生成大量高效的测试输入. 该引擎首先会利用污点分析来收集程序执行过程中的路径列表, 并对其中不重要的路径进行路径缩减, 得到一个缩减后的路径序列用于分析. 之后, Intriguer 的引擎采用一种字

段级别的路径推断技术将路径序列进行字段级别划分和标识. 最后, 该引擎会构建一个字段路径的迁移树, 用于分析该字段的不同分支跳转等. 基于这棵迁移树, 该引擎对字段进行约束的收集求解, 并给出该字段可能的取值用于辅助模糊测试的输入生成. 借助这种办法产生的测试输入可以触及到一些很难进入的分支, 并触发一些极端情况下的漏洞, 可以有效提高密码算法库模糊测试输入的有效性.

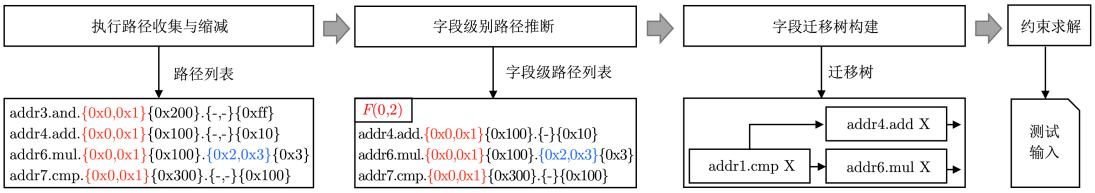


图 8 Intriguer 采用的路径约束求解算法

Figure 8 Path constraint solving algorithm used by Intriguer

虽然目前的典型工具对测试输入的有效性进行了大量的技术研究, 但是目前的技术缺乏对密码算法库本身语义的理解, 导致生成的输入有效性依然不够. 比如 Jin 等人采用的混合符号执行算法, 虽然可以标记追踪某些关键值的路径并收集约束, 但是无法根据不同的密码算法进行不同的关键值标注. 如果可以深入理解不同算法的语义, 再根据语义中所包含的关键变量或者字段进行约束求解, 会更好地提高密码算法库的测试效果.

3.4 多类型漏洞识别技术

密码算法库模糊测试的最后一个关键技术是对多类型的漏洞进行识别的技术. 在测试的过程中, 一个关键的问题是什么时候才算是触发了密码库的漏洞. 根据第 2 节中对密码算法库需求的描述, 需要识别的漏洞类型大体包括三方面: 内存安全、逻辑安全和兼容性安全. 接下来将根据这几个漏洞类型来分别对目前典型工具的支持情况进行介绍.

内存安全. CryptoFuzz、CLFuzz、Jin 的工具和 ecfuzzer 等都支持对内存安全漏洞的检测, 通过对密码算法库进行 ASAN [18] 插桩的方式来检测内存溢出等漏洞. ASAN 是一款针对内存相关漏洞的检测器, 它可以在任何的内存相关操作开始之前对该操作是否存在内存安全问题进行检查, 并发出告警.

逻辑安全. 目前的典型工具都支持对密码算法库的逻辑漏洞的检测, 但是具体的支持手段和类型不一样. 具体来说, CryptoFuzz 支持对内部不一致这种逻辑漏洞的检测. 内部不一致指的是, 同一个密码算法的实现, 在内部数据处理的过程中对不同场景的处理结果出现不一致性. 比如 AES 对称加密, 加密的密文不能被算法的解密函数正确解密. 为了检测这种类型的漏洞, CryptoFuzz 在编写算法驱动的时候, 会对这样的内部逻辑一致性进行检查, 比如让 AES 的加密函数生成密钥, 同时验证解密函数能否完成解密. Jin 等人的工具支持的逻辑漏洞类型和方法跟 CryptoFuzz 一样.

CLFuzz 支持的逻辑漏洞类型是在 CryptoFuzz 上进行扩展的. CLFuzz 利用交叉验证的技术将不一致性检查扩展到了同种密码算法的不同实现上. 比如 OpenSSL 的 AES 算法实现的加密函数得到的密文, 交给 LibreSSL 的 AES 算法实现的解密函数用相同的密钥解密, 观察是否一致. ecfuzzer 支持的逻辑漏洞检测主要是面向椭圆曲线的. 在每一轮模糊测试过程中, ecfuzzer 会检查当前椭圆曲线对两个输入的大数计算出来的标量乘法结果和前一个模型的结果的对比, 如果不一致则认为发现了椭圆曲线计算过程中的漏洞.

CDF 和 WycheProof 则依靠驱动中的测试断言来支持逻辑漏洞的检测. 它们会根据当前密码算法的逻辑来设计一些断言, 比如 CDF 对 DSA 算法实现了检测无限循环的测试断言, 对 RSA-OAEP 算法实现了时间泄漏的测试断言等等. WycheProof 的断言都是从过往的攻击案例中提取出来的, 比如针对椭圆曲线算法的非法曲线攻击、对于数字签名算法的随机数偏好攻击以及 Bleichenbacher 攻击 [27] 等.

兼容性安全. CryptoFuzz、CLFuzz、Jin 等的工具和 CDF 等工具都支持对密码算法库的兼容性漏洞检测, 他们所采用的方法都是差分测试. 即对同一个密码算法的同一个功能, 寻找几个不同的密码算法库的实现来对同样的输入进行处理, 最后比较他们之间的输出. 如果这些实现的输出彼此之间存在差异, 那么一定至少有一个密码算法库对该算法的实现是有漏洞的.

目前的工具虽然支持丰富的漏洞类型, 但是对不同类型的漏洞还存在着误报、漏报等问题. 比如对于内存安全漏洞来说, ASAN 虽然是很有效的检测手段, 但是对内存泄漏等漏洞, ASAN 存在着一定程度的误报, 导致对漏洞的检测结果不够准确. 对于逻辑安全来说, 目前 CryptoFuzz 和 CLFuzz 等工具支持的逻辑漏洞类型较少, 无法满足不同算法的特定逻辑检查需求. WycheProof 和 CDF 虽然可以支持更多的逻辑漏洞, 但是设计这些断言需要很多领域知识和人工成本. 对于兼容性安全来说, 目前的工具所采用的差分测试技术的被测对象需要至少有两个及以上的不同版本的实现. 而且差分测试的主要问题是当发现多个版本的执行结果不同的时候, 只能说明当前某个版本存在漏洞, 不能确定性地找出哪个版本是存在漏洞的. 这个问题在只有两个版本的实现进行差分的时候尤为明显, 但是多个版本一起实现差分测试的时候, 可以采用少数原则来解决这个问题. 后续需要利用更多的漏洞识别手段, 来降低检测过程的误报和漏报, 同时提高漏洞断言设计的自动化水平.

4 典型密码算法库模糊测试工具测评

为了评估目前典型的针对密码算法库的模糊测试工具的效果, 本节基于 4 个常用的密码库, 对本文提到的 6 款典型模糊测试工具进行评估. 由于 Jin 等的工具并未开源, 本文只用 CryptoFuzz、CLFuzz、CDF、WycheProof 和 ecfuzzer 这 5 种工具进行实验评估. 本文实验过程中选取的密码算法库分别是: OpenSSL (3.1.0 版本) 、SymCrypt (100.20.0 版本) 、mbedtls (2.28.0 版本) 以及 Crypto++ (8.7.0 版本). 这些密码库使用广泛, 且实现的方式、目标都不尽相同. 具体来说, OpenSSL 的 GitHub 仓库包含 24 000 多个关注, 它的使用场景非常广泛, 从网络通讯协议到大型的软件系统, 都利用 OpenSSL 来实现密码逻辑. 与 OpenSSL 类似, Crypto++ 也包含丰富的密码算法. Crypto++ 被广泛使用在开源项目、学术研究等非商业化的软件中. SymCrypt 则是微软公司开发的密码算法库, 目前被用在 Windows 平台中进行加密逻辑的处理. 自从 Windows 10 操作系统发布以来, SymCrypt 已经成为了 Windows 上所有密码算法的主要实现库. Mbedtls 则以体量偏小的代码实现各种加密逻辑. 这样做的目的是使这些加密逻辑可以很方便地用于各种嵌入式设备上. 目前, mbedtls 在嵌入式系统中应用十分广泛, 而且实现了丰富的加密逻辑. 总的来说, 在这四个密码算法库上进行实验可以得出一些具有通用性的结论. 本文的实验在一台 64 位的物理机上进行, 机器运行的操作系统是 Ubuntu 20.04.2 LTS, 该机器有 256 G 的内存空间. 为了得到准确的效果, 每个实验均进行了 3 次重复实验, 对结果取平均值用于实验结论的分析. 为了评估目前工具的效果, 本文从三个维度展开分析, 分别是: 漏洞发现能力、测试分支覆盖数量以及测试输入合法比例.

4.1 漏洞发现能力

为了评估各个典型工具的漏洞发现能力, 本文将 5 个典型的模糊测试工具在各个密码算法库上运行了 24 小时, 并统计了这段时间内发现的漏洞数量. 结果如表 2 所示.

表 **2** 典型模糊测试工具在各个密码算法库上的漏洞发现数量

Table 2 Bug numbers of typical fuzzing tools on each cryptographic library

工具名称	漏洞发现数量			
	OpenSSL	**SymCrypt**	**mbedtls**	**Crypto++**
CryptoFuzz	1	0	0	0
CLFuzz	2	2	1	2
CDF	0	0	0	0
WycheProof	0	0	0	0
ecfuzzer	0	0	0	0

经过 24 小时, CLFuzz 是发现漏洞数量最多的工具, 一共发现了 7 个漏洞. 而 CryptoFuzz 在 OpenSSL 上发现了 1 个漏洞, 其他三个工具没有发现任何的漏洞. CLFuzz 发现的 7 个漏洞中, 有 2 个是内存漏洞, 2 个是逻辑漏洞, 3 个是兼容性漏洞. CryptoFuzz 发现的漏洞是内存漏洞. CLFuzz 的漏洞发现能力强主要得益于它对测试输入合法性的保障上, 通过抽取密码算法函数参数的规范信息, CLFuzz

产生的测试输入大部分符合规范, 因此能测试到密码库的深层逻辑中. 虽然 WycheProof、CDF 和 ecfuzzer 也支持合法性的输入生成, 但是 WycheProof 的测试用例相对较老, CDF 的测试场景不多, 而 ecfuzzer 不支持有效的测试输入的生成, 所以导致它们无法发现更多的漏洞.

从以上的数据分析, 总结如下几个发现: (1) 较早的测试工具迄今为止发现的漏洞总数量更高, 但是新的工具发现新漏洞的能力更强. 虽然 WycheProof 等工具包含大量的测试用例, 但是这些用例是由过去的攻击案例总结来的, 对新漏洞的发现没有太大的帮助. 这就需要测试工具高效地动态构造测试输入, 这时有效性和合法性就成为了测试过程能否深入到密码算法库底层逻辑的关键, 新工具往往可以构造更加高质量的输入并发现新的漏洞. (2) 基于断言的逻辑检测器漏洞挖掘能力有限. WycheProof 和 CDF 等基于断言来对逻辑漏洞进行检查, 但是没有发现新的逻辑漏洞. 由于断言插入的粒度比较细, 而且所涵盖的场景比较单一, 所以自动化的逻辑漏洞检查手段往往比简单的断言更加有利于漏洞挖掘.

4.2 分支覆盖数量

为了评估各个工具在不同密码算法库上的分支覆盖数量, 本文将不同库中所包含的密码算法分为三大类: (1) 哈希和对称加密; (2) 椭圆曲线算法; (3) 密钥生成算法. 第一类包含所有对哈希值计算和对称加密、对称解密的算法. 第二类则表示椭圆曲线计算相关的算法. 第三类则代表密钥的生成相关算法. 在运行 24 小时之后, 典型工具在这几类算法上的分支覆盖数量如表 3 所示. 表中 CryptoFuzz 和 CLFuzz 都可以对这三种算法进行测试. CDF 在这几个密码库的支持上并没有支持密钥生成算法, 因此没有统计该算法的覆盖数量. WycheProof 只在 OpenSSL 上支持密钥生成算法, 它支持的其他密码库大部分是 Java 的, 不在此次实验的评估对象上. ecfuzzer 仅支持椭圆曲线算法, 因此实验仅仅评估了它在该算法上的覆盖情况.

表 3 典型工具在不同密码算法上的分支覆盖数量

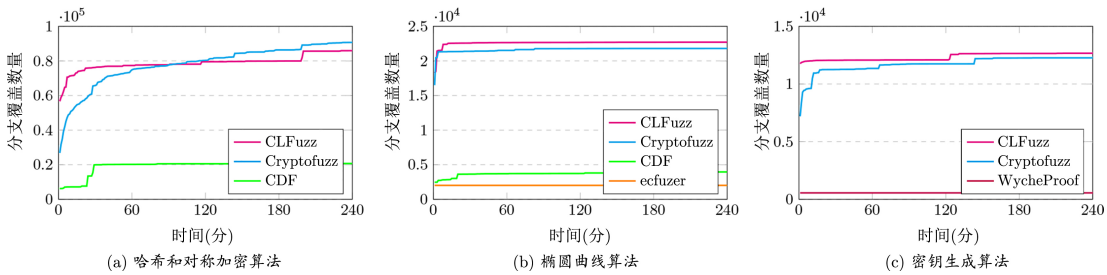
Table 3 Branch coverage of typical tools on various cryptographic algorithms

工具名称	分支覆盖数量		
	哈希和对称加密算法	椭圆曲线算法	密钥生成算法
CryptoFuzz	90 679	21 796	12 264
CLFuzz	85 904	22 707	12 662
CDF	20 714	3960	-
WycheProof	-	-	560
ecfuzzer	-	2012	-

表中的数据表明, 在哈希和对称加密算法上, CryptoFuzz 的覆盖分支数量最高, 这主要是由于哈希和对称加密算法对输入的格式要求相对较少, 因此在固定时间内, CryptoFuzz 可以生成更大量的输入来不断尝试探索新的分支. 在椭圆曲线算法上, CryptoFuzz 凭借其输入合法性和有效性的相关技术而获得了最高的覆盖数量. 在密钥生成算法上, CryptoFuzz 和 CLFuzz 的分支覆盖数量相差无几. CDF 由于输入的生成过程会按照测试人员定义的配置文件进行遍历, 因此测试的效率不高, 覆盖情况也与其他两个工具相差较大. WycheProof 和 ecfuzzer 支持的密码算法数量不够多, 因此覆盖的分支也比较少.

此外, 为了观察这些工具在不同算法平台上的覆盖数量的增长趋势, 实验中还对这些工具的覆盖随时间的变化情况进行了收集分析. 实验数据表明, 这些覆盖数量的变化情况在 4 小时之后趋于稳定, 因此实验统计了 4 小时之内的覆盖数量变化曲线, 结果如图 9 所示. 从图中可以看出, CLFuzz 的覆盖率增长速度很快, 这表明它可以在初期构造出大量的高质量测试输入并快速探索密码算法库的深层逻辑.

总的来说, 目前的典型模糊测试工具可以快速覆盖密码算法库的主要逻辑, 但是覆盖的增长在很短的时间之后就会停止. 这表明目前的模糊测试工具可能会遇到测试瓶颈而无法继续展开深层逻辑的测试. 此外, 不同工具在不同类型的密码算法的测试表现也不尽相同, 比如在哈希和对称加密算法上, 目前的工具可以持续地产生测试输入带来覆盖的增长, 但是在椭圆曲线算法上, 覆盖数量的增长很快就收敛了. 未来在密码算法库的模糊测试技术研究上可能需要着重突破测试过程中无法覆盖的分支, 并分析模糊测试产生的输入无法触发该分支的原因, 并开发更加有效的输入生成技术来提高模糊测试的效果.



(a) 哈希和对称加密算法 (b) 椭圆曲线算法 (c) 密钥生成算法

图 9 典型工具在不同密码算法上的覆盖变化趋势

Figure 9 Coverage trend of typical tools on various cryptographic algorithms

4.3 测试输入合法性

本文列举的典型工具当中, CDF 通过手动配置测试输入, WycheProof 手写了固定的测试用例, ecfuzzer 则固定输入的格式, 这三款工具都可以保证测试输入是完全合法的. 但是 CDF 和 WycheProof 保证合法性的成本很高, 而 ecfuzzer 保证合法性的方式具有局限性, 无法用于非椭圆曲线算法上. CryptoFuzz 和 CLFuzz 都有相关的技术来低成本地保障测试输入的合法性. 为了评估这些技术的效果, 本文对这两个工具在 24 小时之内产生的测试输入的合法输入进行统计, 结果如表 4 所示.

表 4 典型工具在不同密码算法上的分支覆盖数量

Table 4 Branch coverage of typical tools on various cryptographic algorithms

密码算法类别	工具	总输入数量	合法输入数量	合法输入占比 (%)
哈希和对称加密算法	CryptoFuzz	11 034 788	7814 566	70.82
	CLFuzz	29 245 132	26 492 680	90.58
椭圆曲线算法	CryptoFuzz	205 052	156 527	76.34
	CLFuzz	1167 107	1066 716	91.39
密钥生成算法	CryptoFuzz	6421 453	5039 096	78.48
	CLFuzz	32 215 216	29 543 630	91.71

由表中的数据可知, CryptoFuzz 的合法输入占比在 70.82%–78.91% 左右, CLFuzz 的合法输入占比在 90.58%–91.71% 左右. CryptoFuzz 仅仅通过字节流的切分去满足参数调用过程中的合法性, 但是却无法保障在密码算法函数内部的规范检查时对输入参数合法性的要求. 因此 CryptoFuzz 生成的输入的合法概率仅有 70% 左右. 而 CLFuzz 虽然通过语义抽取的技术对测试输入的合法性规范进行了提取, 并在生成测试输入的时候考虑这些规范, 但是却不能达到 100% 的合法概率. 这主要是因为 CLFuzz 对语义的抽取还不够准确和全面, 导致部分规范检查并没有被抽取出来.

总的来说, 目前的典型工具可以通过自动化技术提高产生测试输入的合法性, 但是由于这些技术并不够完备, 导致合法的测试输入比例还有进一步的提高空间. 而通过手动配置等方式满足合法性的技术则存在成本高、应用范围窄等局限性. 因此后续需要继续探索自动化等输入合法性保障技术来提高模糊测试的效率.

5 总结与展望

5.1 密码算法库模糊测试工作的总结

密码算法库被广泛应用于各种大型系统中, 用于保证数据传输过程中的隐私性和安全性. 在密码算法库中的安全漏洞会对系统的稳定运行和数据安全造成严重的影响, 因此对密码算法库中的实现漏洞进行高效的挖掘是学术界和工业界的研究热点. 目前, 已经有许多模糊测试工具针对密码算法库进行漏洞的自动检测, 它们可以对内存安全、逻辑安全和兼容性安全等问题进行自动挖掘. 通过构造测试驱动和大量的测

试输入, 这些工具已经在目前的主流密码算法库 (OpenSSL、SymCrypt 以及 Crypto++ 等) 中发现了大量的漏洞.

但是目前的典型工具对密码库进行模糊测试还面临着如下几个挑战: 一是驱动构造的质量和成本. 由于测试驱动对密码算法库的测试效果影响巨大, 目前的工具都尝试构造大量高质量的测试驱动. 但是由于不同密码算法的驱动内容不尽相同, 目前测试驱动的高质量构造成本比较高. 二是测试输入的合法性保证. 目前的工具为了保证生成测试输入的合法性, 采用手动配置的手段或者是自动化语义抽取的方式, 但是手动配置效率低而且应用范围相对较窄. 而自动化的语义抽取由于准确性和完备性不够, 导致生成的输入合法性仍有提升空间. 三是测试输入的有效性保证. 目前的工具为了保证生成输入的有效性会利用各种边界值和特殊值, 以及符号执行等方式来探索更多的密码算法库深层逻辑. 但是这些方法还是无法高效获取和利用密码算法库的深层语义, 因此有些特殊的密码算法逻辑依然不能被生成的输入覆盖. 四是测试支持的漏洞类型不够. 目前的工具对逻辑漏洞的检测方法还主要依赖人工构造, 缺乏自动化的高效手段.

此外, 在国密算法的支持方面, 目前的密码算法库模糊测试工具没有直接支持国密算法. 但是, 当前的工具均可以对国密算法进行适配, 但是适配的成本不同. 比如, 对于 CryptoFuzz、CLFuzz、ecfuzzer 和 Jin 等人的工具来说, 可以通过实例化模板类的方式, 来支持国密算法的具体实现, 适配成本相对较低, 只需要在模板类中添加国密算法的一些具体的要求规范即可. 对于 CDF 来说, 适配国密算法可能需要找到国密算法的对称加密、签名算法等具体算法的其他实现. 在 CDF 的驱动中, 对这些实现进行调用, 并比较输出结果. 相对而言, 这种适配方式成本较高. 对于 WycheProof 来说, 适配国密算法需要针对算法本身的使用场景进行特殊用例的编写构造, 适配成本是最高的.

5.2 未来展望

未来的密码算法库模糊测试可能有如下几个发展的方向:

(1) **测试驱动的自动构造技术.** 本文介绍了一个高质量的测试驱动需要包含三个方面: 场景丰富、断言多样以及完备的资源管理. 目前也有一些针对库软件的测试驱动自动生成的技术. 未来针对密码库模糊测试的工具需要在这些自动化驱动构造的工具基础之上, 提取密码算法库本身的语义信息和特征, 来分析密码算法的不同场景, 构造不同的断言, 同时控制记录所有申请的资源并在合适的时候进行释放.

(2) **测试输入的合法性保障技术.** 目前的工具在对测试输入进行生成时, 主要是通过手动配置和语义抽取来确保合法性. 为了解决手动配置的局限性以及语义抽取的不准确性, 后面的模糊测试工具需要自动化地对被测密码库进行静态分析, 并尝试精确提取参数规范检查相关的函数, 来获取参数的合法规范. 在生成新的测试输入的时候, 要避免相关变异算法产生的输入违背这些规范.

(3) **测试输入的有效性保障技术.** 目前的工具虽然可以通过符号执行和特殊值填充等方法提高测试输入的有效性, 但是仍然无法深入到某些特殊的密码算法逻辑中, 导致漏洞发现能力相对较低. 未来的模糊测试工具可能可以利用大语言模型或者污点追踪等方式, 对不同密码算法的语义进行深入的理解, 来生成一些可以覆盖关键分支的测试输入.

(4) **多类型漏洞识别技术.** 目前的工具对逻辑漏洞的支持过多依赖手动定义的断言. 未来的模糊测试工具可能可以建立一些自动化的手段来对密码算法的逻辑实现漏洞进行判定. 比如交叉验证、椭圆曲线点坐标检查等方法, 可以借助自动化对程序分析等技术来自动提取验证的目标.

参考文献

[1] DAEMEN J, KNUDSEN L, RIJMEN V. The block cipher Square[C]. In: Fast Software Encryption—FSE '97. Springer Berlin Heidelberg, 1997: 149–165. [DOI: 10.1007/BFb0052343]

[2] JIAO L, HAO Y L, FENG D G. Stream cipher designs: A review[J]. SCIENCE CHINA Information Sciences, 2020, 63(3): 1–25. [DOI: 10.1007/s11432-018-9929-x]

[3] KOBLITZ N, MENEZES A, VANSTONE S. The state of elliptic curve cryptography[J]. Designs, Codes and Cryptography, 2000, 19: 173–193. [DOI: 10.1023/A:1008354106356]

[4] RIVEST R. The MD5 message-digest algorithm[R]. 1992. [DOI: 10.17487/RFC1321]

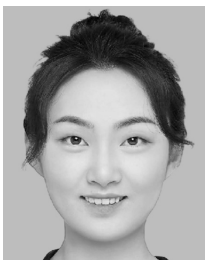
[5] SM2[EB/OL]. Wikipedia. 2024-01-18. https://zh.wikipedia.org/wiki/SM2

[6] SM4[EB/OL]. Wikipedia. 2024-01-18. https://zh.wikipedia.org/wiki/SM4

[7] OpenSSL. Cryptography and SSL/TLS toolkit[EB/OL]. 2024-01-18. https://www.openssl.org/
[8] Microsoft. SymCrypt is the core cryptographic function library currently used by Windows[EB/OL]. 2024-01-18. https://github.com/microsoft/SymCrypt
[9] Crypto++. Crypto++ library is a free C++ class library of cryptographic schemes[EB/OL]. 2024-01-18. https://www.cryptopp.com/
[10] The Heartbleed bug. 2024-01-18. https://heartbleed.com/
[11] National vulnerability database: CVE-2024-0727[EB/OL]. 2024-01-18. https://nvd.nist.gov/vuln/detail/CVE-2024-0727
[12] Google. American fuzzing lop[EB/OL]. 2024-01-18. https://github.com/google/AFL
[13] LLVM. LibFuzzer—A library for coverage-guided fuzz testing[EB/OL]. 2024-01-18. https://llvm.org/docs/LibFuzzer.html
[14] GitLab. Peach[EB/OL]. 2024-01-18. https://peachtech.gitlab.io/peach-fuzzer-community/
[15] VRANKEN G. CryptoFuzz—Differential cryptography fuzzing[EB/OL]. 2024-01-18. https://github.com/guidovranken/cryptofuzz
[16] ZHOU Y H, MA F C, CHEN Y L, et al. CLFuzz: Vulnerability detection of cryptographic algorithm implementation via semantic-aware fuzzing[J]. ACM Transactions on Software Engineering and Methodology, 2023, 33(2): 1–28. [DOI: 10.1145/3628160]
[17] National vulnerability database: CVE-2023-5363[EB/OL]. 2024-01-18. https://nvd.nist.gov/vuln/detail/CVE-2023-5363
[18] SEREBRYANY K, BRUENING D, POTAPENKO A, et al. AddressSanitizer: A fast address sanity checker[C]. In: Proceedings of 2012 USENIX Annual Technical Conference (USENIX ATC 12). USENIX, 2012: 309–318. [DOI: 10.5555/2342821.2342849]
[19] CDF—Crypto differential fuzzing[EB/OL]. 2024-01-18. https://github.com/kudelskisecurity/cdf
[20] Google. WycheProof[EB/OL]. 2024-01-18. https://github.com/google/wycheproof
[21] JIN H, AN D, KWON T. Differential testing of cryptographic libraries with hybrid fuzzing[C]. International Conference on Information Security and Cryptology. Springer Cham, 2022: 124–144. [DOI: 10.1007/978-3-031-29371-9_7]
[22] ecfuzzer: Differential fuzzing for elliptic curves[EB/OL]. 2024-01-18. https://github.com/catenacyber/elliptic-curve-differential-fuzzer
[23] LibreSSL[EB/OL]. 2024-01-18. https://www.libressl.org/
[24] Google. BoringSSL is a fork of OpenSSL that is designed to meet Google's needs[EB/OL]. 2024-01-18. https://github.com/google/boringssl/tree/master
[25] ZHANG M R, ZHOU C J, LIU J Z, et al. Daisy: Effective fuzz driver synthesis with object usage sequence analysis[C]. In: Proceedings of 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, 2023: 87–98. [DOI: 10.1109/ICSE-SEIP58684.2023.00013]
[26] CHO M, KIM S, KWON T. Intriguer: Field-level constraint solving for hybrid fuzzing[C]. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2019: 515–530. [DOI: 10.1145/3319535.3354249]
[27] RONEN E, GILLHAM R, GENKIN D, et al. The 9 lives of Bleichenbacher's CAT: New cache attacks on TLS implementations[C]. In: Proceedings of 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 435–452. [DOI: 10.1109/SP.2019.00062]

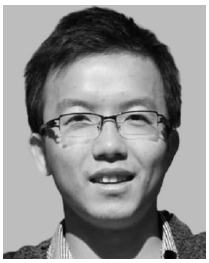
作者信息

马福辰 (1997–)，辽宁沈阳人，博士研究生. 主要研究领域为区块链与分布式系统安全、密码算法库模糊测试.
fuchenma525@gmail.com

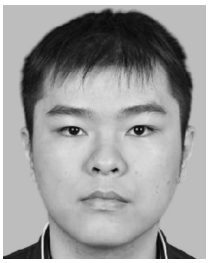


周远航 (1999–)，山东青岛人，硕士研究生. 主要研究领域为密码库模糊测试、区块链安全.
cindyzhouyh@gmail.com

陈元亮 (1995–), 江西南昌人, 博士研究生. 主要研究领域为漏洞挖掘与软件安全分析.
sard.chen@gmail.com



姜宇 (1989–), 江西南昌人, 副教授. 主要研究领域为模糊测试与程序分析.
jiangyu198964@126.com



颜臻 (2001–), 广东广州人, 本科生. 主要研究领域为区块链与分布式系统安全.
yanzhen20011121@163.com

孙家广 (1946–), 江苏镇江人, 教授. 主要研究领域为计算机辅助设计、软件架构及应用.
sunjg@mails.tsinghua.edu.cn