**Semi-valid Input Coverage for Fuzz Testing**

ABSTRACT

定义了模糊测试的第一个覆盖准则半有效输入覆盖（SVCov）。当有效输入可以由一组有限的约束定义时，我们的准则是适用的。SVCov度量测试在多大程度上覆盖了半有效输入的域，其中一个输入是半有效的，只要它满足除一个外的所有约束。

以模糊测试网络密钥交换协议（IKE）为例，说明SVCov的实用价值。研究表明，精确定义和有效测量SVCov是可行的。此外，SVCov为提高模糊测试的有效性和增强模糊测试工具和库提供了感知信息。特别是，通过增加SVCov下的覆盖率，我们在成熟的IKE实现中发现了一个以前未知的漏洞。

1. INTRODUCTION

覆盖标准是测试实践[12]和质量保证标准[26]的组成部分。根据给定的标准测量测试集的覆盖率与暴露被测系统（SUT）中的故障相关：低覆盖率提示测试丢失，并建议如何改进测试集。然而，现有的覆盖标准不适合测量模糊测试的覆盖率。特别是，它们不度量fuzz测试的全部内容，即使用半有效输入执行SUT。

本文提出了一种模糊测试的覆盖准则。在解释为什么需要一个特定于模糊测试的覆盖准则，以及我们提出的准则的特点之前，我们简要回顾了模糊测试的原理。fuzz测试没有正式的定义，而且fuzz测试和其他测试技术之间的界限并不清晰。然而，模糊测试通常被理解为使用SUT规范无法预见的输入执行SUT。然后检查SUT是否存在一般不希望的行为，例如内存访问冲突；参见[19，32]。

用于模糊测试的输入必须是半有效的[22]，即根据规范无效的输入，但也不能完全无效。无可否认，这是一个模糊的定义。关键是输入解析器过滤掉了大多数完全无效的输入输出，因此不要在解析器代码之外运行软件。例如，打算使用jpeg文件的程序不太可能接受pdf文件。然而，程序很可能接受修改过的jpeg文件，其中有一个位已经被翻转，并且在执行过程中抛出一个异常。在本例中，修改后的jpeg文件是半有效输入，而pdf文件不是。本文在半有效输入概念的基础上，定义了模糊测试的覆盖准则。

定义了半有效输入覆盖准则，简称SVCov。当有效输入可以由一组有限的约束定义时，我们的准则是适用的。例如，实现通信协议的软件的有效输入正是那些满足协议规范中定义的所有约束的输入。我们定义一个半有效输入为满足除一个外的所有约束的任何输入。当且仅当对于每个约束c，集T包含至少一个违反c且同时满足所有其他约束的测试时，才认为测试集T完全符合SVCov。

我们注意到，在文献中已经定义和评估了无数的测试覆盖标准，如[7，36]。现有的许多标准也适用于模糊测试。例如，有经验证据表明，增加语句覆盖率可以提高模糊测试在发现错误方面的有效性[32]。现有的标准根据测试执行的语句数量等来衡量一组模糊测试的覆盖率。但是，它们并不衡量SUT在多大程度上使用了半有效的输入。例如，只包含有效输入的测试集可以执行SUT中的所有代码状态；然而，使用这样的测试集测试SUT显然不是彻底模糊测试的实例。与现有准则相比，SVCov度量测试覆盖半有效输入域的程度，从而比其他准则更能反映模糊测试的定义。我们不认为SVCov可以替代现有的标准。相反，与任何其他软件度量一样，SVCov补充和平衡了其他适用的标准，参见[5]。

**贡献**。提出了模糊测试的第一个覆盖准则SVCov。为了显示它的实用价值，我们对SVCov的三个中心要求进行了研究：

**R1 Feasibility可行性：**必须能够（1）在合理的时间内从现实世界的规范中精确定义SUT的半有效输入，（2）有效地测量SVCov。

**R2 Relevance to coverage:与覆盖率的相关性：**测量SVCov必须向测试人员提供关于如何提高测试集覆盖率的有意义的信息。

**R3 Relevance to discovering faults:**R3与发现故障的相关性：增加测试集的SVCov将导致发现SUT中的其他故障（如果有）。

为了判断SVCov是否满足这些要求，我们对网络密钥交换协议（IKE）进行了模糊测试。模糊测试安全协议（如IKE）是公认的挑战[32]，因为安全协议是有状态的，它们有复杂的输入结构，并且使用加密。模糊测试IKE实现的挑战性和IKE规范的规模（包括三个rfc[25，17，13]），使IKE成为我们研究的代表性候选。

我们对IKE的实验证实，通过从现实世界的协议标准中提取约束来精确定义半有效输入集确实是可行的，在IKE的情况下，这些约束跨越了许多rfc。此外，检查测试（用于测量SVCov）约束的可满足性所需的时间平均比执行测试所需的时间小两个数量级。因此，在测试期间测量SVCov的开销可以忽略不计。

我们对SVCov进行了测量，并对结果进行了分析，找出了部分半有效输入在测试中丢失的原因。分析指出了模糊测试设置中的一些微妙问题，并为我们提供了提高测试覆盖率的基本信息。具体来说，我们发现了（1）SecFuzz中的实现缺陷[34]，这是本研究中使用的模糊测试工具，（2）SecFuzz的模糊算子的不精确性，以及（3）我们从IKE的rfc中提取的约束的冗余性。SVCov还指出了缺失的有效输入，这对SecFuzz的有效性和基于变异的模糊测试技术（更多细节见第3.4.3节）至关重要。解决这些问题大大提高了测试的覆盖率。

我们发现在第一个实验中丢失的测试很重要；在第二个实验中，我们在一个流行的、稳定的IKE实现中发现了一个以前未知的漏洞。发现的漏洞与安全相关：攻击者可以利用该漏洞来颠覆IKE。我们已将此漏洞告知软件供应商。

综上所述，我们的案例研究证明了SVCov的测量是可行的，可以用来提高模糊测试工具和库的质量，特别是测试集的覆盖率。此外，我们的测试集的SVCov增量发现了一个以前未知的严重安全漏洞。总的来说，我们与SVCov的初步经验是非常令人鼓舞的；关于我们未来的工作，请参见第5节。

**关于术语的一个词**。通过测试，我们意味着执行SUT并检查其失败行为。失败是错误的表现，是人类在构建SUT时所犯错误的结果；参见[14]。漏洞是与安全相关的错误，攻击者可以利用这些漏洞进行攻击。每个测试（也称为测试用例或测试输入）是一对输入和预期输出。在fuzz测试的上下文中，测试输入是SUT规范无法预见的。也就是说，规范没有规定用于模糊测试的输入的预期输出；因此，它不能用作测试oracle。因此，在模糊测试中使用的测试oracle通常会检查通常不需要的属性。换言之，所有测试输入的预期输出是相同的，例如没有内存访问冲突发生。

测试的目的是发现SUT中的故障。一组测试只要暴露了SUT的所有错误就足够了。这种充分性的定义，尽管到目前为止，几乎没有实际价值，因为尚不知道SUT中的整个故障集（否则，不需要测试）。因此，不能根据这个定义来衡量测试集的充分性。特别是，在测试集没有显示失败的令人不安的情况下，我们不能决定SUT是无故障的，还是测试集根本不充分。

如果且仅当一组测试达到全部（100%）覆盖率时，根据给定的覆盖率标准，该组测试才被视为彻底的。我们注意到彻底性和充分性之间的关系是脆弱的：不彻底的测试集可能会错过重要的测试，尽管不是每个彻底的测试集都足够。

论文提纲。本文的其余部分安排如下。在第2节中，我们正式定义了SVCov并描述了如何实例化和使用它。在第三章中，我们以模糊测试IKE为例说明SVCov的实用价值。我们回顾了相关的覆盖标准，并将其与第4节中的SVCov进行了比较。最后，在第五章中，我们对论文进行了总结，并讨论了我们今后的工作。

2. SEMI-VALID INPUT COVERAGE

我们简要描述了测试覆盖率的概念。接着介绍了模糊测试准则SVCov，并描述了SVCov的具体实现和使用方法。

**2.1 Coverage Criteria Axioms 覆盖标准公理**

文献中定义和研究了大量的测试覆盖标准，如[7，36，20]。覆盖率标准用于衡量测试的进度，并决定何时可以停止测试，或者偶尔自动生成和修剪测试集。但是，必须注意不要使测试过度符合覆盖率标准，因为这可能会导致测试效率降低[27，30]。

覆盖准则必须满足许多基本公理：当测试集被扩大时，空的测试集必须被指定为零覆盖，并且测试集的覆盖率决不能降低。此外，要排除一致地为每个测试集分配零的平凡标准，需要对任何覆盖标准至少有一个完整的测试集。这些要求通常由文献中关于空集的不充分性、单调性和适用性的公理来表述；参见例如[35，36]。

模糊测试使用半有效输入来练习SUT。因此，我们为模糊测试的覆盖标准添加了以下公理：

模糊测试覆盖公理：仅由有效输入组成的测试集实现零覆盖。

这个公理指出，除非无效输入属于测试集，否则测试集不会模糊测试SUT。因此，当涉及模糊测试时，任何只包含有效输入的测试集必须被指定为零覆盖。

基于SUT模型和程序结构的各种覆盖准则（如语句覆盖、分支覆盖、转移覆盖、决策覆盖等）不符合模糊测试覆盖公理是很容易检验的。类似地，基于输入域分区及其边界值的覆盖标准不会显式地区分有效、半有效和完全无效的输入（我们将很快定义这些术语）。它们也不符合模糊测试覆盖公理。这并不奇怪，因为现有的覆盖标准并不适合模糊测试。下面，我们介绍一个符合模糊测试覆盖公理的模糊测试覆盖准则。

2.2 SVCov Coverage

定义了模糊测试的覆盖准则。本节中的定义是抽象的：我们不指定输入域、约束的特征等。在第3节中，我们给出了此处定义的所有概念的具体示例。

我们假设给定了一个有限的非空约束集，使得S定义了手头SUT的有效输入集。有效输入集，表示为Ivalid，是满足S中所有约束的最大输入集，

因此，无效输入至少违反了S中的一个约束。仅违反一个约束的无效输入称为半有效输入。半有效输入被认为对模糊测试特别有效，因为它们可能通过SUT的输入验证过滤器[32]。我们在第5节讨论了半有效输入的替代定义。

应该说几句：

SVCov的定义与用于生成测试的技术无关。例如，SVCov准则可用于测量使用基于变异的技术、使用语法等生成的模糊测试的覆盖率；参见[22，32]。

SVCov准则仅依赖于约束集和测试集，SUT的结构（如源代码）不起作用。从这个意义上说，SVCov补充了引用SUT代码或模型的现有标准。

对于SVCov而言，一个彻底的测试集可能是不充分的。然而，我们认为，根据SVCov，不彻底的测试集会错过能够揭示SUT中错误的测试（见下文）。如前所述，这一观察结果并不特定于SVCov：它适用于所有覆盖标准。

在给定一组约束条件下，SVCov的最大覆盖是可以实现的，即存在一个测试集T，使得SVCov（T）＝1，iffs是最小的。这是因为如果S包含一个冗余约束c，那么σc=Φ。

SVCov提供的信息量取决于S中的约束条件。例如，如果S是一个单例，SVCov相当不具信息性，因为对于任何T，SVCovS（T）的值要么是0，要么是1，不包括中间的值。在我们的案例研究中，我们确认为实际相关的协议定义一组足够详细的约束是可行的，并且事实上是简单的。

我们注意到使用约束集来实现高SVCov的测试集的有向构造是可能的。根据约束的性质，可以自动生成测试用例，例如，一次一个地否定每个约束，并解决否定约束与剩余约束的结果连接。然而，定向测试生成不在本文的讨论范围内，因此这里不作进一步讨论。

我们声称SVCov满足第1条中给出的三个要求，即可行性、与覆盖范围的相关性和与发现故障的相关性。为了验证这一点，我们使用Internet密钥交换协议进行了一个案例研究。在这个案例研究中，我们观察到调查SVCov低的原因可以揭示与模糊测试相关的所有组件中的问题。通过解决这些问题，我们能够通过在SUT中重新发现具体漏洞的测试来扩展我们的测试集。案例研究见第3节。

2.3 Guarded Constraints

我们注意到，定义实际系统的约束通常隐式地定义了输入必须满足的适用性条件。例如，约束“对于任何输入，第三个字节必须是前两个字节的异或”显然只适用于长度大于三个字节的输入。在我们的研究中，我们给出了IKE输入的适用条件示例，见第3节。下面我们介绍保护约束的概念，以说明适用条件。

2.4 Measuring and Using SVCov

在本节中，我们将讨论SVCov的实际方面。我们首先描述了如何测量SVCov，然后讨论了影响SVCov的典型因素。

度量SVCov的第一步是指定约束集，这些约束集共同定义SUT的有效输入。设T为针对SUT执行的测试集。对于T中的每个测试，检查测试违反了多少约束。如果一个测试恰好与一个约束相关，则违反的约束将添加到集合CovS（T）。最后，计算定义1中的SVCovS（T）。算法1给出了这个过程的伪代码。

算法1要求S中的每个约束c是一个递归集：对于每个测试t，它必须是可判定的t∈c。约束在实践中通常是无限集（见第3节），因此是使用其特征函数定义的。也就是说，对于每个c∈S，存在一个可计算函数x2 c，其中，如果t∈c，则x c（t）=1，否则x c（t）=0。特征函数通常解析t，如果t满足某些条件，则返回1。例如，（人工）约束的特征函数“对于任何输入，第三个字节必须是前两个字节的异或”将解析输入，计算前两个字节的异或，然后将结果与第三个字节进行比较。如果它们相等，则函数将返回1，否则返回0。第3节给出了特征函数的程序示例。

在对一组测试的SVCov进行测量之后，对于一组约束，测试人员可以确定T所达到的覆盖率是低的。这个决定通常依赖于SUT的风险分析、测试人员可用的时间和资源等。测试人员必须调查SVCov低的原因，以增加T的覆盖范围。通过我们的案例研究，我们确定了一些影响SVCov的因素：

（1） fuzz测试工具中的实现缺陷和不精确的fuzz操作符。

假设要违反给定的约束，字段中的某个位必须设置为零，而fuzz运算符只能用零替换所有字段的位。在这种情况下，如果用零替换剩余的位会导致违反附加约束，则模糊运算符对于手头的任务是不精确的。我们在第3节中给出了模糊算子不精确的具体例子。

为了解决这个问题，测试人员必须修复模糊测试工具并改进其模糊操作。

（2） 非最小约束集。

很明显，如果约束c是冗余的，则不能仅仅因为σc=∏而唯一地违反约束；见第2.2节。

为了解决这个问题，测试人员必须在S中分析约束，并通过重写或删除约束来避免任何冗余。

（3） 无效输入。

我们已经观察到，通过改变有效输入（真空地满足S中的约束）而创建的测试可能会导致低SVCov。我们在第3.4.3节中回到这一点。

为了解决这个问题，测试人员必须修改生成有效输入的算法/程序。如前所述，这个因素只与基于变异的模糊测试技术有关。

（4） 时间。

模糊测试有可能唯一地违反约束c，但测试不包括σc，因为模糊测试过早停止。

为了解决这个问题，测试人员必须花更多的时间对SUT进行模糊测试。

测试人员应该如何以及在多大程度上解决这些问题取决于可用的时间和资源。测试人员可能决定解决（某些）问题，并重复对SUT进行模糊测试。

SVCov与违反的约束集的比较。为了有一个用于度量SVCov的参考点，我们定义了一个度量，称为冲突，它度量冲突约束的总数。形式上，给定一组约束和一个测试集T，违规定义为

请注意，违犯计数T违犯的所有约束，而SVCov是指唯一违犯约束的数量（见第2.2节）。因此，如果一个测试t∈t同时违反S中的两个约束，则两个约束都被计入违反，但两个约束都不计入SVCov。显然，对于任何S和T，ViolatedS（T）都大于SVCovS（T）。

通过与SVCov的比较，找出SVCov偏低的原因。Violated和SVCov之间的差距表明，要么模糊算子是不精确的，即它们“降级”有效输入，使多个约束被违反，要么S是非最小的；参见上面的因子（1）和（2）。从未违反的约束条件表明，有效输入是空的（见第2.3节），或由于超时而从未生成某些半有效输入；见上文因子（3）和（4）。

违反的计算算法与算法1相似，因此在此省略。

3. SVCov FOR FUZZ TESTING IKE

我们的案例研究旨在探讨测量SVCov的可行性和效益。我们的SUT是OpenSwan[23]，它是Internet密钥交换协议（IKE）的成熟开源实现。安全协议的实现，例如IKE，对fuzz测试来说是一个挑战[32]，因为它们有复杂的输入结构，使用加密，并且是有状态的。

这一节的其余部分组织如下：在第3.1节中，我们描述了我们的实验设置，其中我们提供了SecFuzz的背景知识，SecFuzz是研究中使用的fuzz测试工具。在第3.2节中，我们给出了提取IKE约束的过程和启发式方法，以及一些具体的例子。在这里我们还讨论了SVCov如何解释IKE的状态性质。我们使用约束来度量SecFuzz生成的测试的SVCov，并在第3.3节中总结了结果。我们在第3.4节中讨论了SVCov低的原因。在第3.5节中，我们介绍了我们的改进，并报告了先前未知的漏洞。

3.1 Experimental Setup

在我们的实验中，我们使用SecFuzz[34]，这是我们之前为模糊测试安全协议实现而开发的工具。SecFuzz使用一个具体的协议实现来生成有效的输入；我们将这个实现称为相反的端点。SecFuzz使用一组fuzz运算符对有效输入进行变异，然后将变异的输入发送到SUT。检查SUT在测试期间的行为是否失败。

如图2所示，SUT是参与安全协议的端点之一，而另一个端点是另一个端点。例如，如果SUT是协议的发起方（通常称为Alice），则相反的端点是响应方（通常称为Bob）。注意，这两个端点不必属于协议的同一软件实现。

我们将通信环境配置为通过SecFuzz路由发送给SUT的消息。因此，相反的端点生成有效的输入并将其传递给fuzzer。SecFuzz的作用是对消息进行变异并将它们转发给SUT。但是，SecFuzz不能直接处理它接收到的消息，因为它们通常是加密的。事实上，对加密消息进行变异肯定会导致垃圾，即SUT会直接丢弃的数据。因此，我们让另一个端点使用日志文件与SecFuzz共享其加密信息（例如密钥、加密和解密库），以允许SecFuzz在对消息进行变异之前对其进行解密。SecFuzz的详细描述见[34]。

在我们的案例研究中，SUT是OpenSwan v2.6.35中的IKE响应器实现[23]。OpenSwan是一个稳定的开源Internet协议安全（IPsec）实现，可用于许多主要的企业Linux发行版。OpenSwan是用C编写的，有629173行代码。SUT在动态分析工具Memcheck[18]中执行，Memcheck[18]是我们的测试oracle。MEMCHECK是基于ValgReND（21）的C和C++二进制程序的内存错误检测器。它检测一系列内存错误，如使用未定义的变量、无效的内存访问、不正确的堆内存管理、内存泄漏等。内存错误检测器被广泛用作模糊测试的测试存储库（参见，例如，[8，12，32]），因为内存错误是安全关键的，并且经常被攻击者利用[33]。

我们使用OpenSwan的initiator作为SUT的相反端点来生成输入。在协议运行中生成的输入取决于另一个端点的配置文件；这些文件引用不同的协议设置。为了确保另一个端点使用一系列不同的配置，我们让另一个端点使用不同的配置文件执行多次，并自动生成配置文件。

安全协议实现（如我们的测试主题OpenSwan）的输入是在协议运行期间交换的消息序列。每个消息通常由多个有效负载组成，其中每个有效负载由多个字段组成。协议标准指定如何解释字段的位。协议实现输入的分层结构允许SecFuzz在不同的抽象级别上改变输入。SecFuzz的fuzz运算符如表1所示。我们注意到消息通常在其字段之间具有依赖关系。例如，长度可变的字符串字段可能有一个专用字段指示字符串的长度。类似地，有效负载通常有一个字段指示消息中下一个有效负载的类型。消息通常在接收时由包过滤器进行预处理，并丢弃字段不一致的消息。为了确保不是所有的变异消息都被丢弃，Sec-Fuzz可以在应用field/payload Fuzz操作符之后选择更新依赖字段；有关更多详细信息，请参见[34]。

3.2 IKE Constraints

IKE是一种广泛部署的安全协议，用于在两个端点之间建立安全关联。安全关联（SA）是一组加密属性（例如加密算法和哈希函数）和一个用于保护信息的安全策略。IKE使用内部安全关联和密钥管理协议（ISAKMP），这是一个用于身份验证和密钥交换的框架[17]。ISAKMP规范定义了在IKE中使用的有效载荷格式（例如提议、密钥交换和标识有效载荷）。艾克分两个阶段进行。第一阶段有两种不同的模式，主模式和攻击模式，这两种模式都在端点之间建立安全关联。协商的SA为端点之间的进一步通信建立安全通道。第二阶段称为快速模式，其目的是代表另一个服务（如IPsec）设置SA。完成第二阶段后，两个端点可以交换附加消息，例如检查另一个端点是否活动。这些消息被指定为信息模式交换。IKE协议有几个RFC文档中指定的多个版本。对于我们的案例研究，我们使用[13]中指定的IKEv1。

要测量一组IKE模糊测试的SVCov，首先必须使用IKE的RFC文档指定定义IKE有效输入的约束集。为此，除了IKEv1（RFC2409[13]），我们还使用ISAKMP协议（RFC2408[17]）的IKE相关子集以及ISAKMP的Internet IP安全解释域（RFC2407[25]）。RFC文档是非正式的，通常很长，例如RFC2408是86页。我们使用标准关键字must，must not，and required作为分析IKE的rfc的启发式方法。这些关键字不仅指向规范中的重要句子，而且往往有一个明确的解释，这使得它们很适合指定约束。IKE约束的示例如表2所示。例如，约束c2引用自RFC2408。

使用上述简单的过程，我们从前面提到的RFC中提取了217个IKE约束，每个主模式、攻击模式、快速模式和信息模式的约束数分别为82、68、55和12。提取约束所需的时间约为8人小时。这些约束是公开的。

接下来，我们从IKE相关rfc中提取的非正式约束被建模为保护约束。例如，约束c2（在表2中）仅适用于包含建议有效载荷的输入。我们将c2表示为cg◃ct，其中cg由包含建议有效载荷的所有输入组成，ct由建议有效载荷的下一个有效载荷字段设置为2或0的所有输入组成。IKE约束中的保护通常检查约束是否适用于特定的IKE交换模式（快速、主或攻击模式），或特定的有效负载（例如SA付费负载、建议有效负载、密钥交换有效负载等），或特定的消息顺序（例如表2中的c1）。

注意，要彻底模糊测试有状态的sut，需要生成半有效的输入，以深入研究系统。以IKE为例，要对协议的第二阶段（即快速模式）的实现进行模糊测试，必须成功地完成协议的第一阶段。实际上，涉及快速模式的保护约束仅适用于成功完成IKE第一阶段的输入。为了违反这样的约束，测试需要完成IKE的第一个阶段，然后违反约束的目标。因此，SVCov的概念解释了IKE中的状态：没有深入研究SUT的测试不会违反引用quick模式（或任何后续模式，为此原因）的约束。

为了自动检查约束，我们使用Scapy对它们进行了编码，Scapy是一个用于解析和操作消息的Python库[28]。图3显示了一个示例实现。图3中的方法检查接受两个参数：消息和消息前面的消息序列。第二个参数是必需的，因为某些约束引用以前交换的消息（在同一协议运行期间）。我们实现了一个约束检查器，它接受一个IKE测试，即一个IKE消息序列，并输出违反的、真空满足的和非真空满足的IKE约束集。

我们测量了用i7-2600四核处理器和8GB内存检查机器约束所需的时间。平均来说，约束验证器需要41mil-liseconds来检查一个IKE输入的所有217个约束，即每个约束0.19毫秒。相比之下，执行单个测试（运行Memcheck时）平均需要1秒。因此，测量覆盖率的开销可以忽略不计。

我们的结论是，在IKE的上下文中，SVCov满足第1条中给出的要求R1：在合理的时间内，精确定义任何IKE实现的半有效输入集是可行的。此外，用于检查约束的可满足性的over-head可以忽略不计。注意，提取定义IKE有效输入集的约束是一次性的任务：这些约束既不绑定到IKE的任何特定实现，也不绑定到用于（模糊）测试IKE的任何特定技术。

3.3 SVCov Measurements

我们使用第3.2节中定义的约束条件（以下简称为SIKE）来测量使用SecFuzz生成的36000个测试的半有效输入覆盖率。下面给出的测量值对应于运行SecFuzz 10小时。也就是说，执行每个测试平均需要1秒。

在图4所示的图中，实线显示SVCovSIKE（T），虚线显示ViolatedSIKE（T），两者都是在SUT上执行的测试数的函数。图中显示，最初由SecFuzz生成的测试会快速增加SVCov。正如预期的那样，随着模糊测试的进行，增加的速度降低了。这是因为最初集合σc没有被覆盖，并且测试很可能覆盖它们。随着测试覆盖的σc越来越多，新生成的测试越来越可能落入已经覆盖的σc，因此它们不会增加SVCov。

实验结束时，SVCov在217个约束条件中占90%，即41%。此外，86%的IKE约束至少被违反一次，因此14%的约束在锡克从未被违反。在第3.4节中，我们解释了SVCov为何较低以及SVCov与违例之间存在较大差异的原因。

在这个实验中，我们在SUT（OpenSwan的响应程序）中发现了一个严重的安全漏洞，我们在CVE-2011-4073中报告了这个漏洞。该漏洞平均有4800个测试。我们注意到模糊测试本质上是随机的。因此，为了测量SecFuzz发现上述漏洞所需的平均时间，我们已经执行了多次测试。

3.4 Coverage Analysis

在这一节中，我们分析了在第3.3节中报告的实验中实现的覆盖范围。分析的目的是了解SVCov为何较低，并找出提高SVCov的方法。

SVCov的定义取决于约束集SIKE和测试集。反过来，SecFuzz生成的测试集依赖于相反端点提供的有效输入集和SecFuzz中实现的fuzz运算符集。因此，影响SVCov的因素有三：（1）在SecFuzz中实现的fuzz算子集；（2）约束集SIKE；（3）有效输入集。下面，我们将讨论与我们发现的每一个因素相关的问题。当然，我们分配给模糊测试IKE的时间也起到了一定的作用。

*3.4.1 Implementation Bugs in* SecFuzz *and Impre- cise Fuzz Operators*

如前所述，SVCov与实验中的冲突有很大的差距。SecFuzz的fuzz算子可能是原因：这些算子“降级”了有效的输入，以至于变异的输入不能唯一地违反SIKE中的某些约束。在实验中，我们以违反但不是唯一的约束为指导，手工研究了SecFuzz的fuzz算子，发现了几个问题：

Scapy库：给定的IKE负载指定加密属性，SecFuzz不能修改这些属性，因为它依赖Scapy。因此，如果约束要求此有效负载指定特定的加密属性，则SecFuzz无法生成唯一绕过约束的测试。这是SecFuzz从Scapy库继承的一个限制。

插入有效载荷运算符：由于其随机性，Sec-Fuzz的插入有效载荷Fuzz运算符并不总是插入根据IKE格式良好的有效载荷。例如，为了违反IKE约束“包含安全关联有效负载的消息不能包含任何标识有效负载”，必须使用SecFuzz的insert payload操作符。然而，插入的识别有效载荷具有压倒性的概率病态。因此，使用SecFuzz生成的测试不太可能唯一地违反约束。

实现错误：我们在SecFuzz中发现了许多实现错误。例如，对于某些运算符，可以放置在字段中的随机数限制为小于100的正整数，而要违反某些IKE约束，SecFuzz必须插入大于100的数字。

总之，以SVCov为指导，我们指出了SecFuzz的fuzz算子中的一些微妙问题。原则上，通过测试fuzz操作符可以发现其中的一些问题（fuzz操作符测试仪也需要测试）。然而，测试操作符仅限于揭示实现中的错误，而不能发现缺少的fuzz操作符或精度问题。

*3.4.2 Non-minimal Set of Constraints*

对于任何约束c，如果σc为空，显然不能覆盖它；请参见第2.2节。通过对SIKE中约束条件的分析，我们得到了一个约束条件c，其中σc=Φ。约束c声明：第一轮消息中的ID字段必须设置为零。此约束包含以下两个约束的交集：

cp:第一轮消息必须包含密钥交换负载。

cq:在包含密钥交换负载的消息中，ID字段必须设置为零。

如果违反c，则消息是第一轮消息，并且ID字段未设置为零。但是，如果消息不包含密钥交换负载，则违反cp；如果它确实包含密钥交换负载，则违反cq。

我们没有从锡克中去掉c，因此可以在第1个实验和第2个实验之间进行公平的比较，如第3.5节所述。注意，上面讨论的重叠约束本身并不是SIKE的问题：它们表明rfc中的相应短语是多余的。

我们注意到，真空满足保护约束cg◃ct的有效输入不适合生成属于σcg◃ct的半有效输入。假设输入i（即i̸∈cg）对约束是真空满足的。让我们为SecFuzz发送给SUT的变异输入编写ˆi。为了使i属于σcg◃ct，SecFuzz的fuzz算子应该“增强”i以满足cg，同时它们应该“降级”i以违反ct。这显然超出了fuzz操作符的用途。实际上，在我们的案例研究中，使用满足cg的有效输入生成的测试总是唯一地违反了79%的约束条件σcg◃ct。

作为有效输入质量的指标，我们可以测量所有有效输入所满足的约束的数量。注意，我们测量的是有效输入的真空度，即由运算端点创建的输入；我们不测量测试的真空度，即SecFuzz发送给SUT的变异输入。我们发现22个约束（10%）在所有有效输入下都是真空满足的。这表明我们的实验缺少有效的输入。

缺少有效输入。22个约束中有18个是真空满足的，因为由相反端点生成的输入在消息中具有固定的有效载荷顺序。例如，携带数据以生成会话密钥的密钥交换有效负载永远不会作为消息中的最后一个有效负载。但是，根据IKE的规范，IKE消息中的有效负载没有排序限制（在快速模式下更改的消息除外）。然而，另一个端点的实现（OpenSwan的发起者）决定有效负载的顺序，这导致18个约束总是空洞地满足。

其他4个约束是真空满足的，因为端点总是使用其IPv4地址进行标识。因此，类似于：IPv6类型的标识负载必须指定有效的IPv6地址的约束总是空洞地得到满足。回想一下，生成的有效输入取决于用于初始化相对端点的配置文件（见第3.1节）。我们注意到，在我们的实验中，没有配置文件指定与IPv4不同的标识类型。例如，没有IPv6和ASN.1 X.500可分辨名称的配置文件。

为了生成不完全满足SIKE中约束的有效输入，我们实现了一个消息预处理器，在将消息中的有效负载交给SecFuzz之前对其进行洗牌，并且我们为另一个端点创建了额外的配置文件，以使用不同的标识类型生成有效输入。这些修改确实是足够的：在改进之后，有效输入没有空洞地满足任何约束。

*3.4.4 Coverage Analysis Summary*

我们仔细分析了实验结果。在SVCov的指导下，该分析在SecFuzz、其fuzz运算符和其预处理库方面带来了一些显著的改进。此外，它还帮助我们发现了RFC本身的冗余。事实上，SVCov为我们提供了查明上述问题的方法，并指导我们修复这些问题。我们通过指出在模糊测试IKE的上下文中满足了第1节中给出的要求R2来结束这一讨论。在下一节中，我们将看到这些改进如何转化为使用更高SVCov对IKE进行更有效的模糊测试。

3.5 Measuring SVCov After Improvements

在解决了SecFuzz中的实现缺陷（除了SecFuzz中的insert payload fuzz操作符，因为它的修复是非平凡的）和缺少有效输入的问题之后，我们重复了fuzz测试IKE的实验。然后像之前一样测量SVCov，如图5所示。新的测量结果表明SVCov从41%提高到89%。也就是说，217个约束中有193个被唯一违反。注意SVCov的增加并不是由于测试集的扩大：在这两个实验中，我们有36000个测试。

就违反情况而言，217个约束条件中有216个被测试破坏。因此，违反了23个约束，但不是唯一的。对于其中的11个约束，总是与附加约束一起违反它们的原因是insert payload fuzz操作符的精度有限。由于锡克语中的重复出现，不能唯一违反一个约束条件；参见第3.4.2节。由于超时，未生成其余约束的相应半有效输入：我们在10小时后停止了实验。

一个弱点。模糊测试的目的是发现SUT中的故障，而不是增加SVCov本身。在改进SecFuzz之后进行的实验中，我们在OpenSwan的响应程序中发现了一个以前未知的安全漏洞。该漏洞平均在34000个测试中发现，它涉及在接收到带有引用ASN.1 X.500 Distin-guished名称的ID负载的攻击模式消息时响应程序中的无效读取。当存储标识的字段被省略时，OpenSwan将访问未分配的内存。未分配的内存访问很危险，因为攻击者很可能利用它们进行攻击[33]。我们已经向OpenSwan开发团队通报了此漏洞的详细信息。

暴露问题的测试是由设置为空字段fuzz运算符生成的。生成测试用例所需的有效输入是带有使用可分辨名称的ID有效负载的攻击性模式消息。在我们之前的实验中未发现该漏洞的原因是，创建测试所需的有效输入不是由相反的端点生成的；请参阅第3.4.3节。

该漏洞由唯一违反SIKE中约束的测试输入揭示。这提供了支持需求R3的证据：根据SVCov增加模糊测试覆盖率会暴露SUT中的更多错误。

4. RELATED WORK

我们不知道任何现有的覆盖标准，具体到模糊测试。下面，我们将SVCov与应用于模糊测试的覆盖标准进行比较。我们同意迈尔斯等人的观点。要生成健壮的测试集，必须使用多个覆盖标准来平衡它们各自的弱点[20]。也就是说，我们将SVCov视为现有覆盖准则的补充。

基于程序的覆盖标准，如语句、决策和条件覆盖（如参见[7、20、36]）被用来衡量模糊测试的彻底性。例子包括[4，10，31]。与这些度量相比，SVCov对程序结构是不可知的；它度量测试覆盖半有效输入域的程度。

文献中也研究了使用输入域分区定义的基于输入的覆盖标准及其边界值；参见，例如，[20，36]。这些标准类似于SVCov，因为它们也基于输入而不是SUT的内部结构来测量覆盖率。然而，现有的基于输入的准则并不考虑半有效输入。例如，Kaksonen和Takanen讨论了一个覆盖度度量，它反映了输入结构的哪些部分已经被模糊测试所改变了[15]。此度量基于输入，但不考虑半有效输入。类似地，Alrahem和Harris根据SUT的规范划分输入域，然后从每个分区中选择fuzz测试[2]。但是，有效、半有效和完全无效输入之间的区别在它们的分区中并不明显。另一个例子是，朱等人。提出了一种对came模型图进行变异的数据变异算子[29]。它们定义输入的有效性约束，并度量有多少测试违反了这些约束。与SVCov相比，它们不区分违反唯一约束的测试和违反多个约束的测试。这种区别对于模糊测试是必不可少的。

Takanen等人。[32]讨论基于（1）它们检测已知漏洞的能力，（2）支持协议的数量，以及（3）使用模糊器的历史故障检测成功估计的预期缺陷计数来比较模糊器的质量保证度量。这些指标与SVCov正交。作者还建议使用协议rfc来构建“否定的”模糊测试。我们不使用rfc来生成测试；相反，SVCov测量这种负模糊测试的覆盖率。

Opstad等人提出了一项关于使用不同工具对250多个解析器进行模糊测试的综合实证研究[24]。他们的目标是找到启发式方法来决定何时停止模糊测试。他们没有定义覆盖标准。

先进的模糊测试技术，如白盒模糊测试（如[6,9,11,12]）、基于模型的模糊测试（如[1,3,16]）和基于变异的密码工具模糊测试（如[34]）是对软件系统进行模糊测试的不同方法。SVCov定义了如何测量使用这种模糊测试方法生成的测试集的覆盖率。

5. CONCLUSIONS AND FUTURE WORK

我们提出了半有效的输入覆盖准则SV-Cov来衡量模糊测试的覆盖率。在对网络密钥交换协议（IKE）进行模糊测试的实例研究中，我们发现SVCov是一种易于测量的方法，覆盖测量开销可以忽略不计。我们的案例研究也证明SVCov是一个信息量大的工具，可以指导测试人员提高模糊测试的覆盖率。此外，我们对模糊测试工具和库进行了改进，以增加SVCov，从而在一个流行的、稳定的IKE实现中发现了一个以前未知的漏洞。

总的来说，我们与SVCov的初步经验是非常令人鼓舞的。接下来，我们计划将我们的实验扩展到一组更大的安全相关协议，以便能够对SVCov的值得出具有统计意义的结论。

在推广SVCov的过程中，可以将半有效输入的定义扩展到违反n个约束中k个的定义。在这篇文章中，我们重点讨论了k=1。类似地，对于每个约束c，完全测试集可以定义为每σc至少包含j个半有效输入的测试集。本文主要讨论j=1。我们打算在安全协议的背景下研究这种扩展的成本效益比。

尽管SVCov的定义与生成测试所用的技术不可知，但将SVCov应用于不同的模糊测试技术将是有趣的；例如，测量由白盒模糊器或基于模型的模糊器生成的测试的覆盖率。这是一个很有意思的课题，因为我们期望增加测试SVCov所需的活动，例如，一个白盒模糊器，与本文描述的活动是不同的。