基于分解重构的网络软件测试数据生成方法

李 程 魏 强 彭建山 王清贤
(国家数字交换系统工程研究中心 郑州 450002)

摘 要 协议测试能够有效检测网络应用软件的缺陷,但是在面对加密和验证机制时,现有方法难以有效构造测试数据。为此,提出一种基于"分解-重构"的网络软件测试数据生成方法,即使用检查点以及解密内存定位技术,结合加密和验证机制的组合情况,分解出测试端未经编码的有效测试数据;借助基于副本消除的内存回溯算法,在通信另一端定位编码前非副本内存,重构编码后测试数据包。实例分析和对比测试表明,该方法能够有效生成测试用例。

关键词 fuzzing技术,污点分析,加密机制,验证机制,符号执行
中图法分类号 TP393.08 **文献标识码** A

Network Software Test Data Generation Based on Decomposition and Reconstruction

LI Cheng WEI Qiang PENG Jian-shan WANG Qing-xian
(National Digital Switching System Engineering & Technological Research Center, Zhengzhou 450002, China)

Abstract Protocol fuzz testing can effectively detect vulnerabilities in network software, whereas when facing encryption and checksum mechanisms, existing approaches are hard to generate valid test data. A test case generation method based on "decomposition and reconstruction" was proposed. By means of detection technology on check point and decrypted memory, the valid decoded test data was decomposed at test side. A memory-backtracking algorithm was also proposed, which detects the memory none of the duplication of other memories at the other side, based on which the encoded test packet is reconstructed. Case study and comparison test demonstrate that the method can effectively generate test cases.

Keywords Fuzz testing, Taint analysis, Encryption mechanism, Authentication mechanism, Symbolic execution

1 引言

随着Web的广泛应用,网络通信安全得到越来越多的关注,网络应用软件的安全性也面临巨大挑战。为保证通信的私密性与安全性,很多网络软件的通信协议采用了加密和验证机制,但这也阻碍了对网络软件的脆弱性测试。目前,网络软件的测试数据大多通过直接篡改通信数据包而生成,典型的有基于中间人篡改[1]、模板生成[2]和协议格式解析[3,4]等测试方法。但是当网络软件存在加密和验证机制时,这些方法存在测试失效的问题。一方面,加密机制会隐藏通信数据的原有语义信息,导致直接篡改数据包难以控制解密后内容。另一方面,验证机制的存在使得直接篡改的测试数据由于完整性检查不通过而被丢弃,导致测试失效。

针对上述问题,研究人员也提出了如下一些解决方法。

第一类是基于实时构造的测试数据生成方法。该类方法需要获知加密协议的实现细节,比较有代表性的是SecFuzz[5]和文献[6]。SecFuzz根据配置的加密算法、密钥等测试信息构造加密数据包,但需要测试人员预先获知加密算法的实现细节。文献[6]通过从软件中提取出加解密函数的二进制形式接口,以中间人方式实时解密通信数据包,对其进行篡改,并再次加密进行重放。但从二进制程序中分离出能独立调用的加解密算法是很困难的工作,受限于密钥等额外参数的获取。

第二类是基于检查点绕过的测试方法。该方法主要用于识别和绕过验证机制的检查点。Taintscope[7]针对文件内部的校验机制,通过对比正常和畸形的测试样本的路径差异性定位程序检查点,并借助符号执行技术[8,9]重新计算攻击样例的校验值。Cui等人[10]通过静态分析的方法定位检查点,并符号化非校验区域以展开测试。这两种方法都受限于符号执行的计算能力,且无法应对加密机制。

第三类是基于符号执行分割求解的方法。为解决符号执行技术难以求解编码函数的问题,该方法将程序中由编码机制影响的约束问题分解,分别对各部分约束求解,最后联合求解结果生成一个完整的输入。典型代表为J. Caballero等人的BitFuzz[11]。但BitFuzz需要提取加密函数的逆函数调用接口以重新编码测试数据,在通信包含非对称密钥的情况下,该方法由于无法根据解密函数特征定位相应加密函数而失效。

以上方法虽然在一定程度上能够缓解加密机制和验证机制带来的问题,但都存在不同程度的局限性:1)需要获知加解密算法的实现细节,在加密算法未知或难以从二进制程序中抽取可独立调用的加解密函数接口的情况下,现有方法不适

到稿日期:2013-01-05 返修日期:2013-03-26 本文受国家高技术研究发展计划(863计划)项目(2012AA012902)资助。

李 程(1989-),男,硕士生,主要研究方向为网络信息安全,E-mail:richardlicheng@163.com;魏 强(1979-),男,副教授,硕士生导师,主要研究方向为网络信息安全;彭建山(1979-),男,讲师,主要研究方向为网络信息安全;王清贤(1960-),男,教授,博士生导师,主要研究方向为网络信息安全。

用。2)无法应对非对称加密机制,包括数字签名等完整性认证机制。

本文提出一种基于"分解-重构"的网络软件测试数据生成方法,即通过定位编码数据分解后的解码内容,对其应用传统测试技术,得到测试信息,并借助于通信另一端重新构造新的编码数据。由于仅关注相关内存的识别与定位,该方法无需获取加解密算法的具体实现,且能够有效应对非对称密钥机制。

2 基于分解重构的测试数据生成思想

为方便描述,全文将加密和验证机制统称为编码机制,并将经过加密或哈希运算的数据统称为编码数据。本小节首先给出一个具体的实例,据此提出本文的模型。

2.1 问题描述

Zeus是一款远控软件,其源码2010年发布于网络上,其中控制端通信过程中的数据处理片断如图1所示。

```
1. HEADER * header=(HEADER *)buffer;
2.   if(rc4Key != NULL)
3.   {
4.     Crypt::RC4KEY rc4k;
5.     Mem::_copy(&rc4k, rc4Key, sizeof(Crypt::RC4KEY));
6.     Crypt::_rc4(buffer, FULL_SIZE_OF_OVERLAY, &rc4k);
7.   }
8.   if(header->magicDword != MAGIC_DWORD ||
     Crypt::crc32Hash((LPBYTE)buffer+OFFSETOF(HEADER,
     dataSize))) !=header->crc32)
10.  {
11.    return false;
12.  }
13.  Mem::_copy(buffer, (LPBYTE)buffer+sizeof(HEADER),
header->dataSize);
14.  return true;
```

图1 Zeus代码片段

从Zeus的数据包处理过程可以看出,控制端首先会对接收数据进行解密(第6行),随后计算数据包数据部分的哈希值(第8行),并与数据包头部的校验值(header->crc32)进行比较。如果值不相等,则函数直接返回错误(第11行),否则对解密后数据进行拷贝(第13行),函数正常返回。因而,直接篡改网络接收数据不仅不能有效控制解密后数据内容,且会由于数据完整性检查不通过而直接被丢弃。

2.2 模型建立

下面给出包含编码机制的C/S通信过程的描述。

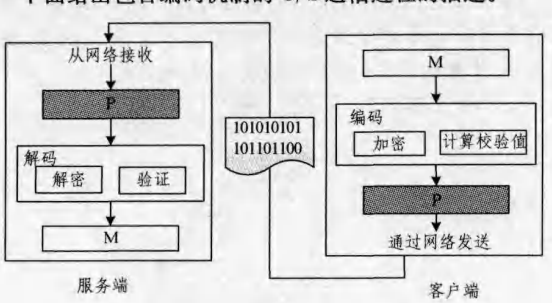

图2 客户端与服务端通信示意图

由图2可知,客户端组织完命令数据M后,对其进行编码,并将数据P发送给服务端。服务端对P进行解密和验证等解码操作,得到未编码数据M,随后对M展开数据格式解析等工作。可以看出,在服务端直接对P进行测试会导致测试失效,而对M进行测试才能达到测试效果。本文称M为有效测试数据。

假定服务端为测试端,"分解-重构"的测试流程为:

(1) 分解:在测试端,根据编码数据P定位解密并通过验证的有效测试数据M。

(2) 测试:对M进行测试,得到相关测试信息。如通过符号执行技术,根据反馈结果得到需要变异的字段。此外,其它方法如基于污点分析的方法[12]也适用于该思路。

(3)重构:根据测试端定位的M,结合测试反馈的变异字段,重新构造经过编码后的测试数据P。

第(2)步的测试方法已经比较成熟,在此不再赘述。下面重点就分解(第3节)和重构(第4节)两方面展开论述。

3 编码数据分解

数据从编码到解码的分解过程,需要考虑两方面的问题:(1)如何识别加密和验证机制,即如何定位解密内存以及检查点;(2)当加密和验证机制同时存在时,如何从中分解出有效测试数据。

3.1 关键定位方法

3.1.1 检查点定位

检查点是程序内部的验证机制用于检查数据完整性的一条特殊指令,典型的检查点指令类型有cmp,sub等。文献[7]针对文件内部检查点机制做了详细论述,本文将其定位方法应用到网络应用软件上,效果良好。具体为:1)跟踪所有条件跳转指令的跳转情况,通过分析正常数据包和畸形数据包的跳转行为差异性,定位跳转分支点。2)从分支点开始,生成后向切片,定位第一个影响EFLAGS寄存器的指令为检查点,记为$Cmp(A,B)$。

3.1.2 解密内存定位

本文提出一种基于依赖度的解密内存定位方法。为便于描述,首先给出如下定义。

定义1(编码函数,Encoding function) 程序内部存在的压缩、加(解)密、哈希等函数调用。为方便描述,用Dec表示解密函数,Enc表示加密函数,$Hash$表示哈希函数。

定义2(依赖源,Data Dependency Source,DDS) 在污点分析[13]过程中,单字节单元x与污染源B中数据相关的字节集合称为x相对于B的依赖源,记为$Src_B(x)$。

定义3(依赖度,Data Dependency Degree,DDD) 在污点分析过程中,单字节单元x与污染源B中数据相关的字节数称为x相对于B的依赖度,记为$DDD_B(x)$或$x \overset{n}{\sim} B$。

例如:对于指令mov eax,ebx,x为eax最低字节,B为ebx,则x相对于B的依赖源为B的第4字节,且$x \overset{1}{\sim} B$。

定义4(多字节单元的依赖源) $A=a_1a_2\cdots a_n$为n字节单元,A相对于污染源B的依赖源$Src_B(A)$为A中每个字节依赖源的并集,即$Src_B(A)=\bigcup_{i=1}^{n} Src_B(a_i)$。

定义5(多字节单元的依赖度) $A=a_1a_2\cdots a_n$为n字节单元,对于污染源B,$A \overset{n}{\sim} B$表示A相对于B的依赖度为n,且$A \overset{n}{\sim} B \Leftrightarrow DDD_B(A)=\min\{DDD_B(a_i)\mid i\in[1,n]\}$。

定义5说明,多字节单元的依赖度就是其依赖度最低字

节的依赖度。

定义 6(强依赖关系) 设置阈值 T 表示依赖度上限。对于依赖关系 $A \overset{n}{\sim} B$,当 $n>T$ 时,称 A 强依赖于 B,记为 $A \propto B$。

定理 1 高依赖度具有传递性,即 $A \propto B, B \propto C \Rightarrow A \propto C$。

根据多字节单元依赖度为依赖度最低字节的特性,结合污点传播的传递性,易证明该定理。

由定义 6 可知,解密数据与密文之间具有强依赖性,这是由于大多数加密算法会频繁混合数据以达到混淆效果,使得解密内存中每一个字节都与密文的多个字节数据相关。

结合以上特性,为了更精确地定位解密内存,设定如下约束条件:

约束 1 解密内存强依赖于加密的网络输入数据。

约束 2 解密内存在编码函数调用过程中生成,且未被覆盖或释放。

该条件限制了解密内存生成的时机,以定位最早生成的解密内存。其中,编码函数的定位采用文献[14]中基于指令类型统计的定位方法。

约束 3 解密内存在编码函数调用完毕后会被读取。

该条件防止将编码函数调用过程中所写的临时高依赖度内存误报为解密内存。

约束 4 解密内存没有用于检查点的对比操作。

该条件防止将哈希类型编码函数所输出的校验值误报为解密内存。

3.2 分解方法

分离出有效测试数据,能够绕过编码函数的阻碍,同时为编码数据的重构做准备。如图 3 所示,分解过程就是根据输入 I 定位有效测试数据 M 的过程。根据验证和加密两种机制的特点,可能存在几种主要编码机制及其组合方式,需要分别判定。

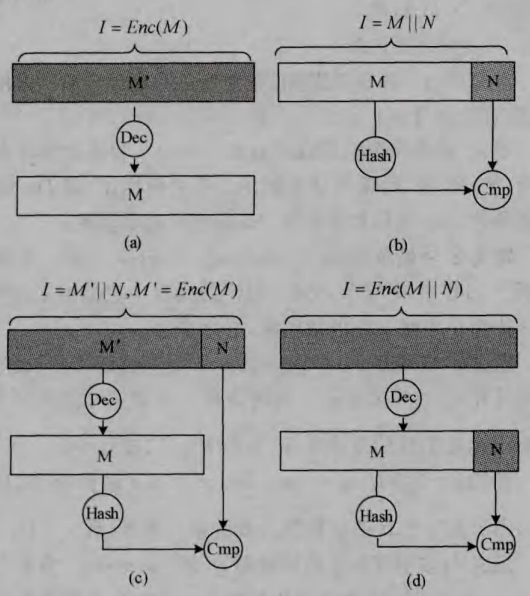

图 3 常见的编码机制及其组合

(1)仅加密,如图 3(a)所示。

$I=Enc(M)$,则定位的解密内存就是 M。

(2)仅验证,如图 3(b)所示。

$I=M||N$,检查点为 $Cmp(N, Hash(M))$。以 I 为污染源,根据检查点的比较情况,有 $Hash(M) \propto M, N \overset{1}{\sim} N$,由此可判定高依赖度区域的依赖源是有效测试数据 M,即为 $Src_I(Hash(M))$。

(3)验证与加密同时存在。

a) 数据被加密,校验域为加密前数据的校验值,如图 3(c)所示。类似于图 3(b),根据检查点的比较情况,有 $Hash(M) \propto M \propto M', N \overset{1}{\sim} N$,可区分高依赖度区域的依赖源是加密后的数据域 M',又根据 $M'=Enc(M)$,可判定解密内存就是有效测试数据 M。

b) 数据与校验域一同被加密,如图 3(d)所示。此情况较为特殊,在以输入 I 为污染源的情况下,$Hash(M) \propto M \propto I$,$N \propto I$,即 $Hash(M)$ 和 N 都强依赖于 I,无法对其进行有效区分。如再一次将解密后的数据视为污染源,即以 $I'=M||N$ 为污染源,有 $Hash(M) \propto M, N \overset{1}{\sim} N$,因而能够判定高依赖度区域的依赖源 $Src_{I'}(Hash(M))$ 就是有效测试数据 M。

综上分析,总结分解方法:如果不存在检查点,则为图 3(a)的情况,解密内存就是有效测试数据。否则,根据检查点比较指令的依赖度进行区分:如果是高依赖度与低依赖度比较,且未定位到解密内存,则是图 3(b)的情况,否则为图 3(c)的情况;如果是高依赖度与高依赖度的比较,则是图 3(d)的情况,需进行二次数据流分析以分离有效测试数据。

4 编码数据重构

编码数据的重构就是要重新构造能够正确解密且能够通过完整性验证的测试数据。本文借助于客户端完成数据的重构,能够有效弥补已有方法的弊端。

4.1 重构思想

重构思想为:既然编码数据包是由客户端(通信另一端)发送,那么客户端程序一定包含了所有的编码功能,因此可以借助于客户端程序完成数据的再次编码工作。需要逆向分析客户端发送数据的执行轨迹(Trace),通过对数据流回溯分析,定位未编码内存,篡改相关字段,借助客户端重新生成编码后的测试数据包。

因此,定位未经编码的数据是关键所在,需要准确把握定位时机。一方面,定位的内存不能过于向上,因为可能数据还未组织完毕。另一方面,也不能过于向下,因为可能定位的内存只是用于临时计算的副本。

例如,图 4 表示了编码过程中的内存组织关系,Mem A 是最终组织完毕的待编码的内存,Mem B 是其一个副本,用于计算哈希值,虽然 Mem B 与 Mem A 的内容相同,但若更改 Mem B,则只会影响计算得到的哈希值,而不会影响发送数据包。同样,更改 Mem C 虽然能够修改发送内容,但由于错过了哈希函数计算而导致附带的哈希值不正确。因而,更改 Mem A 才能从源头上保证生成有效编码数据。

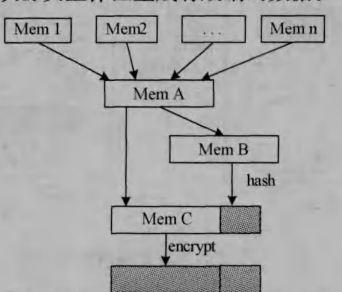

图 4 编码数据生成过程示意图

加密和验证机制的特点是在待发送的命令数据组织完毕后，才完成最后的编码工作，利用该特点可以解决定位过上问题。通过定位编码函数，从其所读的缓冲区中找到与服务端有效测试数据相同的内存，称这些内存为候选缓冲区。

为解决定位过下问题，需要从候选缓冲区中回溯到第一个不是从其他地址复制过来的缓冲区，称其为源缓冲区，如图4中的Mem A。显然，源缓冲区是通过一定的计算或组织得到的，如：从Unicode字符串转化为ANSI字符串，或者是从不同分散内存集中而成的内存。

4.2 基于副本消除的内存回溯算法

内存回溯算法的目的就是从候选缓冲区出发，从执行轨迹中逆向找到源缓冲区。首先设计单字节回溯算法。

算法1 单字节回溯算法

输入：最近一次写该字节的指令 a

输出：二元组(i,b)，i 为回溯到的最后一条指令，b 为源操作数类型，b∈{IMM,MEM,REG,UNKNOWN}

```
0. ξ={mov,xchg,push,pop,bswap,stos}
1. TraceSource(inst a){
2. if(Instruction type of a is not in ξ)
3.      return(a,UNKNOWN)
4. t=type(src(a))//指令 a 的源操作数类型
5. if(t==RGE){
6.    Find nearest list a1 where dst(a1)==src(a)
7.    return TraceSource(a1);
8. }
9. else if(t==MEM)
10.    return(a,MEM)
11. else if(t==IMM)
12.    return(a,IMM)
13. }
```

图5 单字节回溯算法

图5描述了对单字节写操作的回溯算法。由于只是为了消除非副本内存，在实现上仅逆向跟踪移动类型指令的执行过程，如mov、push、pop等指令(第0行)，而在遇到其余类型指令时停止回溯，本文将这些类型指令的源操作数定义为未知(如add等运算指令，第3行)。显然，当一个写操作的源操作数为立即数或未知时，则停止回溯(第3行、第12行)。此外，回溯是一个递归的过程，当内存A的源包含内存B时，可以将内存B视为新的候选缓冲区再进行回溯，因而，在源操作数为内存时也停止回溯(第10行)。

例如，图6表示了缓冲区A每个字节的回溯过程，每一个方块代表一个写操作，包含了写指令地址(Instr)、类型(Type)、源操作数(Sloc)、目的操作数(Dloc)。其中第一个字节经历了3次回溯操作后，最后一次写操作表明写该字节的源为立即数。对于第二字节，由于其最后的写操作来源于add指令，因而其写来源为未知。而写Mem(A+2)到Mem(A+10)的源就是Buffer B。

显然，如果写A的源是内存B，则A就是B的副本，需再对内存B进行回溯。反之，则认为A是源缓冲区。

综上，在单字节回溯算法基础上，基于副本消除的内存回溯算法如下：

Step 1 根据文献[14]定位客户端调用的所有编码函数。

Step 2 提取编码函数调用过程中所读的内存集合 B_1，B_2，…，B_n，从中找到 B_i，其内容与服务端分解过程中定位的有效测试数据 M 相等。若存在多个 B_i 满足条件，选择最早读取的 B_i 为候选缓冲区以减少第三步的回溯次数。

Step 3 对 B_i 的每个字节，用单字节回溯算法找到其写来源的集合，记为 B_t，如果 B_i 与 B_t 内容相等，则将 B_t 视为新的源进行回溯，重新执行Step 3。否则，B_i 就是最终的源缓冲区。

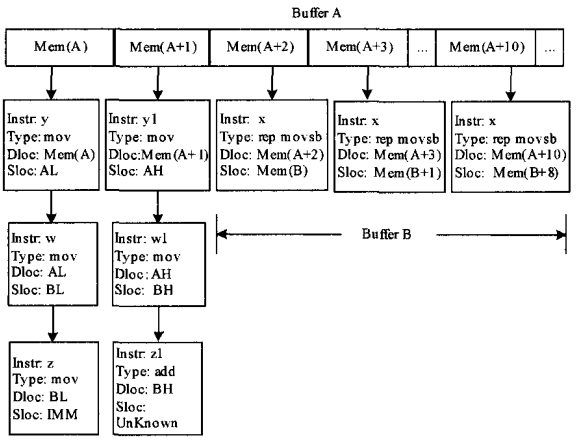

图6 单字节回溯过程

5 实验与分析

为验证方法的有效性，本文构建了实验环境，选取真实网络应用软件进行分析与验证。

5.1 实验环境

实验环境为：Inter i7处理器，4GB内存，1TB硬盘。操作系统为Windows 7和Ubuntu 11.10。

在二进制分析平台BAP[15]上，开发系统EncTracer。EncTracer在跟踪程序处理数据的过程中，能够生成执行轨迹(Trace)，同时集成了污点分析、符号执行等功能模块。

5.2 分解与重构能力验证

IRC协议是一种即时通信协议。为了保证通信的私密性，很多IRC服务器采用了加密机制。本文选取IRC服务器软件UnrealIrcd-SSL(版本为9.2)以及客户端软件mIRC(版本为7.27)为实验对象，二者通信采用了SSL加密机制。在客户端输入命令/join #channel1后，跟踪服务端对数据的处理过程。

5.2.1 分解能力验证

在定位解密内存时，为了防止阈值过大而导致漏检，设置依赖度阈值为5。根据解密内存定位的约束条件1和2，首先定位两个编码函数所写的4块高依赖度内存，其中第一块内存(0xbeb9ff1)和第三块内存地址(0xbeba000)相邻，其依赖度分别为0x10和0x18，这是由于每一轮解密都依赖于上一轮的解密结果，因而依赖度逐渐增加。其后，结合约束3和约束4条件，最终定位了一块解密内存(0xbeb9ff1)，定位过程如图7所示。

此外，EncTracer定位了位于0x5bf963处的检查点，如图8所示，且依赖度对比符合图3(d)的情况，因而判定校验域经过加密处理。将解密数据再次作为污染源进行分析，得到如下结果：

0x5bf963:cmp edx,[ecx]

HighDDS:[0,0x0e],LowDDS:[0x0f,0x22]

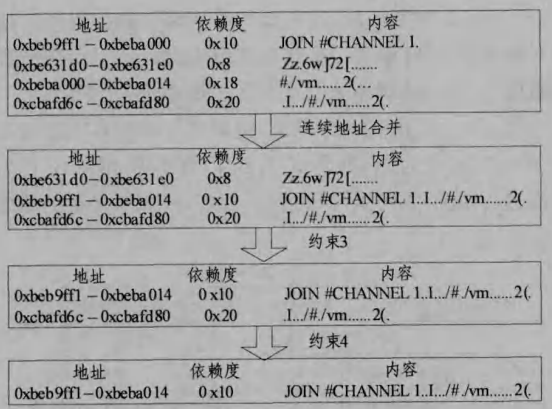

图 7 解密内存定位过程

表示高依赖度的数据来源为解密内存的前 15 字节，即为 JOIN ＃CHANNEL1.，低依赖度的依赖源为解密内存的后 20 字节，为. I…/＃./vm……2(.。因而最终分解得到解密内存的前 15 字节为有效测试数据。

```
005BF95E mov edi, edi
005BF960 mov edx, dword ptr [ebp]
005BF963 cmp edx, dword ptr [ecx] //检查点
005BF965 jnz 005BFACD //分支点
005BF96B sub eax, 4
005BF96E add ecx, 4
005BF971 add ebp, 4
005BF974 cmp eax, 4
005BF977 jnb short 005BF960
```

图 8 检查点的反汇编代码片段

5.2.2 重构能力验证

对客户端的 Trace 进行离线解析，在定位的两个编码函数中，定位其读取的位于 0x210bfa0 和 0x143124b 处的两个内存与有效测试数据相同，将它们标记为候选缓冲区。

对两个候选缓冲区分别应用内存回溯算法，还原内存组织关系，如图 9 所示。可以看出，两个候选缓冲区都是 0x31549c1 处内存的副本，而该内存是由两部分缓冲区拼接而成（由指令①和②完成）的，因而定位 0x31549c1 处内存为源缓冲区。

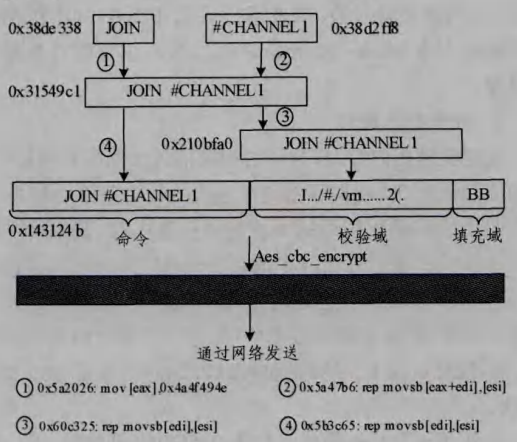

图 9 客户端内存组织关系

为验证内存定位的有效性，修改源缓冲区中的一个字节，将生成编码数据包发送到服务端。经跟踪发现，该数据包能够通过服务端完整性验证，且解密后内存与客户端所修改内存一致，证明了重构方法的有效性。

5.3 测试有效性验证

为验证本方法对于测试有效性的提升，选取两种方法进行对比测试。方法 1：参考文献[8]，采用传统动态符号执行技术，直接将接收的网络数据进行符号化。方法 2：利用本文的方法，仅符号化有效测试数据，并进行重构。二者的路径遍历算法均采用基于代的遍历算法[8]；且设置约束求解器 STP[16] 的求解时间上限为 5 分钟，超过时间上限则放弃该路径分支的求解。

在实现上，方法 2 在解密和验证阶段关闭符号执行模块，进行实际执行，在解密完毕后，将有效测试数据视为符号值，进行符号执行。此外，为了提高测试效率，修改检查点的 EFLAGS 寄存器标志位，强制数据通过完整性验证。

为加快测试进度，方法 2 对客户端采用内存快照[17] 的方式，在有效测试数据组织完毕后进行内存快照，并在下一次测试进行快照恢复，恢复到初始状态。

表 1 为方法 1 和方法 2 在有限时间（120 分钟）内的基本块（Basic Blocks，BBLs）[18] 覆盖数的对比。

表 1 基本块覆盖数对比表

程序（协议）	编码类型	基本块（BBLs）覆盖数		
		方法 1	方法 2	提升率
Apache（HTTPS）	加密＋校验，组合方式如图 3(d)	63443	105945	66.9%
Feiq	仅加密，组合方式如图 3(a)	4532	5217	15.1%
UnrealIRCd（IRC）	加密＋校验，组合方式如图 3(d)	12284	19522	58.9%
PCShare	仅压缩，组合方式如图 3(a)	4743	8355	76.2%
DesDemo	加密＋校验，组合方式如图 3(c)	1224	2102	71.7%

可以看出，在相同时间内，本方法相比方法 1 能够有效提高代码覆盖程度，这是因方法 1 遇到编码函数不能在有效时间内求解所致。本方法虽然需要进行数据重构，但相对于约束求解器的求解时间而言，其时间开销可以忽略。相对于方法 1，本方法对 PCShare 的基本块覆盖提高达到 76%，而对 Feiq 的提高只有 15%，这是由于 PCShare 将其所有通信数据做了加密处理，而 Feiq 仅将会话数据和部分附带信息进行加密，因而对代码覆盖的提升不明显。

此外，通过对 Apache 和 UnrealIRCd 的通信分析发现，二者在密钥交换的通信过程中采用了非对称加密机制，因而文献[5-7]和文献[10,11]的方法都不再适用；而本文由于仅跟踪定位关键内存，而不陷入编码函数的实现细节，因而同时适用于包含非对称密钥机制网络软件的测试数据生成。

结束语 本文提出一种基于“分解-重构”的网络软件测试数据生成方法，其能够有效解决已有协议测试技术在面对加密机制和验证机制存在的测试失效问题。该方法不需要获知编码函数的实现细节，且适用于包含非对称加密机制软件的测试数据生成。需要说明的是，全文将测试目标设为服务端进行论述，但方法同样适用于以客户端为测试目标。

本文方法的前提条件就是需要获取通信双方二进制形式的软件，在一方软件无法获取的情况下（如 QQ 服务器软件等），本方法受限。此外，由于需要在客户端程序运行过程中动态修改内存，对于一些包含反调试功能的软件，本方法同样不适用。以上问题今后将做进一步的研究。

参 考 文 献

[1] ProxyFuzz [EB/OL]. http://www.darknet.org.uk/2007/06/proxyfuzz-mitm-network-fuzzer-in-python/

[2] SPIKE Proxy[EB/OL]. http://www.immunitysec.com/resou-

rcesfreesoftware
[3] 李伟明,张爱芳,刘建. 网络协议的自动化模糊测试漏洞挖掘方法[J]. 计算机学报,2010,34(2):242-255
[4] Milani C P, Gilbert W, Christopher K, et al. Prospex: protocol specification extraction[C]// Proc. of the 30th IEEE Symposium on Security and Privacy. Oakland, California, USA, 2009:110-125
[5] Tsankov P, Dashti M T, Basin D. SECFUZZ: Fuzz-testing security protocols[C]// Proc. of the 7th International Workshop on Automation of Software Test(AST). Zurich, Switzerland, 2012
[6] Caballero J, Johnson N, McCamant S, et al. Binary code extraction and interface identification for security applications[C]// Proc of the 16th ACM Conference on Computer and Communications Security(CCS). Chicago, USA, 2009
[7] Wang T, Wei T, Zou W. TaintScope: a checksum-aware directed fuzzing tool for automatic software vulnerability detection[C]// Proc. of the 31st IEEE Symposium on Security & Privacy (S&P). Oakland, California, USA, 2010
[8] Godefroid P, Levin M Y, Molnar D. Automated whitebox fuzz testing[C]// Proc. of the 16th Network and Distributed System Security(NDSS). California, USA, 2008
[9] 过辰楷,姬秀娟,许静. 基于分支混淆算法的符号执行技术[J]. 计算机科学,2012,39(9):115-119
[10] Cui Bao-jiang, Ji Yu-peng, Wang Jian-xin. An instruction-level symbolic checksum system for windows x86 program[J]. Chinese Journal of Electronics, 2012, 21(1):23-26
[11] Caballero J, Poosankam P, McCamant S. Input generation via decomposition and re-stitching: finding bugs in malware[C]// Proc. of the 18th ACM Conference on Comput Communications Security(CCS). Chicago, USA, 2010
[12] Ganesh V, Leek T, Rinard M. Taint-based directed whitebox fuzzing[C]// Proc. of the 31st International Conference on Software Engineering. Vancouver, Canada, 2009
[13] Kang M G, McCaman S, Poosankam P, et al. DTA++ dynamic taint analysis with targeted control-flow propagation[C]// Proc of the 18th Annual Network and Distributed System Security Symposium(NDSS). San Diego, California, USA, 2011
[14] Felix G, Carsten W, Thorsten H. Automatic identification of cryptographic primitives in binary programs[C]// Symposium on 14thRecent Advances in Intrusion Detection (RAID). Menlo Park, California, 2011
[15] Brumley D, Jager I, Avgerinos T, et al. BAP: The CMU binary analysis platform[C]// Proc. of the 23rd Conference on Computer Aided Verification(CAV). Snowbird, UT, 2011
[16] Ganish V, Dill D. STP: A decision procedure for bit-vectors and arrays[C]// Proc. of the 19th International Conference on computer Aided Verification. Berlin, Germany, 2007
[17] In Memory Fuzzing [EB/OL]. https://www.corelan.be/index.php/2010/10/20/in-memory-fuzzing/
[18] PIN-A Dynamic Binary Instrumentation Tool [CP/OL]. http://www.pintool.org

(上接第 103 页)

[2] 张鹏,崔勇. 移动自组织网络路由选择算法研究进展[J]. 计算机科学,2010,37(1):10-38
[3] Royer E M, Chai-Keong T. A Review of Current Routing Protocols Ad-hoc Mobile Wireless Networks[J]. IEEE Personal Communications, 1999, 6(2):46-55
[4] Raju J, Garcia-Luna-Aceves J J. A Comparison of On-Demand and Table Driven Routing for Ad-Hoc Wireless Networks[C]// Proc of IEEE Int Conf On Communications. Piscataway, NJ: IEEE, 2000:1702-1706
[5] Rahman K K U, Zaman Rafi U, Venugopal R A. Performance Comparison of On-Demand and Table Driven Ad-hoc Routing Protocols using NCTUns[C]// Proc of 10th Int Conf On Computer Modeling and Simulation. Piscataway, NJ: IEEE, 2008: 336-341
[6] Tseng Y-C, Ni S-Y, Chen Y-S, et al. The Broadcast Storm Problem in a Mobile Ad-hoc Network[J]. Wireless Networks, 2002, 8(2/3):153-167
[7] Dai F, Wu J. An Extended Localized Algorithm for Connected Dominating Set Formation in Ad-hoc Wireless Networks[J]. IEEE Transactions on Parallel and Distributed Systems, 2004, 15(10):908-920
[8] Xie Rong, Qi De-yu, Li Yong-jun, et al. A novel distributed MCDS approximation algorithm for wireless sensor networks [J]. Wireless Communications and Moblie Computing, 2009, 9 (3):427-437
[9] Garey M R, Johnson D S. Computers and Intractability: a guide to the theory of NP-Completeness[M]. New York, USA: Freeman, 1990
[10] Dube R, Rais C D, Wang K Y, et al. Signal stability-based adoptive routing(SSA) for Ad-hoc mobile networks[J]. IEEE Personal Communication, 1997, 4(1):36-45
[11] Toh C K. Associativity-Based Routing for Ad-Hoc Mobile Networks[J]. Wireless Personal Communications, 1997, 4(2):103-139
[12] 吴大鹏,武穆清,甄岩,等. 面向链路稳定性的 MANET 路径建立机制[J]. 电子与信息学报,2009,31(9):2226-2231
[13] 胡曦,汪晋宽,王翠荣. MANETs 稳定性路由的移动自适应策略研究[J]. 计算机学报,2011,34(1):96-104
[14] Perkins C E, Royer E M. Ad-hoc on-demand distance vector (AODV) routing[C]// Proc of IEEE 2nd Workshop on Mobile Computing Systems and Applications. Piscataway, NJ: IEEE, 1999:90-100
[15] Marina M K, Das S R. On-demand multipath distance vector routing in Ad-hoc networks[C]// Proc of IEEE 9th Int Conf on Network Protocols. Piscataway, NJ: IEEE, 2001:14-23
[16] Broch J, Jetcheva J, Johnson D B. The Effects of On-Demand Behavior in Routing Protocols for Multihop Wireless Ad-hoc Networks[J]. IEEE Journal on Selected Areas in Communications, 1999, 17(8):1439-1453
[17] Bai F, Sadagopan N, Krishnamachari B, et al. Modeling Path Duration Distributions in MANETs and Their Impact on Reactive Routing Protocols[J]. IEEE Journal on Selected Areas in Communications, 2004, 22(7):1357-1373
[18] Han Y, La R J, Makowski A M, et al. Distribution of path durations in mobile Ad-hoc networks——Palm's theorem to the rescue[J]. Computer Network, 2006, 50(12):1887-1900
[19] La R J, Han Y. Distribution of Path Durations in Mobile Ad-hoc Networks and Path Selection[J]. IEEE/ACM Transactions on Networking, 2007, 15(5):993-1006
[20] Perkins C E, Belding-Royer E M. AODV-PA: AODV with path accumulation[C]// Proc of IEEE Int Conf on Communications. Piscataway, NJ: IEEE, 2003:527-531
[21] Johnson D B, Maltz D A, Hu Y C. The dynamic source routing protocol for mobile Ad-hoc networks(DSR)[EB/OL]. 2012-11-14, http://www.ietf.org/rfc/rfc4728.txt