ISSN 1000-0054 清华大学学报（自然科学版） 2017 年 第 57 卷 第 9 期 2/19
CN 11-2223/N J Tsinghua Univ (Sci & Technol), 2017, Vol. 57, No. 9 903-908

基于脆弱点特征导向的软件安全测试

欧阳永基[1]， 魏 强[1]， 王嘉捷[2]， 王清贤[1]

（1. 解放军信息工程大学 数学工程与先进计算国家重点实验室，郑州 450002；2. 中国信息安全测评中心，北京 100085）

摘 要：为克服模糊测试方法具有盲目性和覆盖率不高的缺点，缓解当前符号执行方法所面临的空间爆炸问题，该文提出一种基于脆弱点特征导向的软件安全测试方法。该方法结合模糊测试和符号执行方法的特点，针对缓冲区溢出，精确分析了具备该脆弱点特征的代码，并以此为测试目标，力图提高测试针对性；通过域收敛路径遍历策略生成新测试数据进行测试。实验数据表明：该方法的缓冲区溢出可疑点识别率比现有的以经验为主的识别方法至少提高 41%，与 CUTE 符号执行工具相比，较好地缓解了空间爆炸问题，并有效验证了 OpenSSL 等常用软件的脆弱点。

关键词：软件安全；特征导向；域收敛；空间爆炸

中图分类号：TP311.1 **文献标志码**：A

文章编号：1000-0054(2017)09-0903-06

DOI：10.16511/j.cnki.qhdxxb.2017.26.038

Guided software safety testing based on vulnerability characteristics

OUYANG Yongji[1], WEI Qiang[1],
WANG Jiajie[2], WANG Qingxian[1]

(1. State Key Laboratory of Mathematical Engineering and Advanced Computing, The PLA Information Engineering University, Zhengzhou 450002, China;
2. China Information Technology Security Evaluation Center, Beijing 100085, China)

Abstract: Fuzzy testing software is random with low coverage while symbolic execution can result in the explosion of the variable space. This paper presents a guided software safety testing method based on vulnerability characteristics that combines fuzzy and symbolic execution. This study analyzed the codes associated with buffer overflow for use as targets to make testing more targeted. Then, new test data was generated using the path traversal patterns of domain convergence. Tests show that the identification rate for potentially vulnerable buffer overflows is at least 41% better than with fuzzy testing, the space size explosion with CUTE greatly reduced with vulnerabilities in common software products such as OpenSSL accurately identified.

Key words: software security; characteristic guided; region convergence; space explosion

在软件安全测试中，Fuzzing 测试是一种广泛采用的方法，它通过向目标系统提供非预期输入并监视异常结果来发现软件漏洞。然而，尽管 Fuzzing 测试部署简单，但它存在盲目性的固有缺陷，为此，研究者提出了许多优化策略，但效果有限[1-6]。为了提高测试的精确性和覆盖率，基于符号执行的测试技术逐渐成为软件测试研究的热点之一。KLEE[7]、CUTE[8]、S2E[9]等是其中的典型代表。利用符号执行技术进行测试，路径遍历是其中一个重要的问题。以 SAGE[10]为代表的测试工具通过取反路径条件，试图遍历所有可行路径，虽然能提高测试覆盖率，但是由于程序的复杂性，将产生空间爆炸的难题，在有限的计算资源下，测试很难适用于大型程序。崔宝江等[11]为了缓解空间爆炸的问题，提出了关键代码区域覆盖的测试方法，虽然减少了无效路径遍历，但是该方法对于关键代码区域的定义过于笼统。Haller 等[12]在符号执行和污点分析的支持下，以数组为测试目标，能够在 Nginx 等复杂的软件下发现程序错误，但是它依赖软件源码，针对目前大多数软件无法获得源码的情况，该方法将使用受限，并且该方法针对每一个可疑候选代码区域都重新执行一次符号执行，可能导致重复执行。

本文提出基于脆弱点特征导向的软件安全测试方法，通过总结缓冲区溢出（buffer overflow，BOF）的特点，刻画脆弱点的二进制形式，以域收敛遍历策略进行导向测试，该方法改善了根据经验方式定义的非确定性，提高了测试目标的针对性，并有效缓解了路径空间爆炸问题。

收稿日期：2016-07-03

基金项目：国家“八六三”高技术项目(2012AA012902)

作者简介：欧阳永基(1985—)，男，博士研究生。

E-mail：oyyj07@gmail.com

1 基于特征导向的软件测试方法

为了促进深层次软件脆弱点的发掘，本文在分析已有缓冲区出错原因的基础上，总结缓冲区溢出的特征，获取可疑的敏感代码空间[13]，以此为待测目标，设计了一种基于域收敛的可疑代码遍历策略，力求提高软件路径遍历的针对性。在该方法的基础上，以缓冲区溢出问题为例进行研究，设计了如图 1 所示的测试框架。

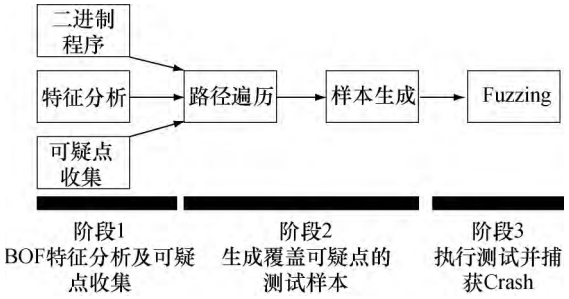

图 1 测试框架

上述测试框架主要由 BOF 特征分析及可疑点收集、样本生成和 Fuzzing 3 个部分构成。其中，BOF 特征分析及可疑点收集主要负责对 BOF 可能的表现形式、特征等进行精确的描述，然后根据获得的信息对二进制代码进行分析，收集可能出现问题的代码，作为待测目标；样本生成通过域收敛遍历策略，促使程序不断覆盖阶段 1 获得的可疑代码区域，并生成对应的测试样本，作为 Fuzzing 输入对象；Fuzzing 主要包括执行测试和 Crash 捕获 2 个主要功能。

2 脆弱点特征分析与可疑点收集

为了克服当前敏感点选择的不足，本文通过对 BOF 脆弱点的分析，发现当前 BOF 的产生一般可以归为 2 类：一是在对内存进行拷贝时，没有进行边界检查，导致了 BOF 的产生；二是在进行内存拷贝时，虽然进行了边界检查，但是由于比较的数据能被控制，特殊的输入也能导致 BOF。

2.1 无边界检查 BOF 特征

CVE-2014-1761 是最近在 Microsoft Office 2010 发现的一个对 RTF 文件进行解析的 BOF 脆弱点，该脆弱点具体出错位置如图 2 所示，在 31D131FF 处获得目标地址时，没有对目的索引指针进行检查，在调用 memcpy 时发生了错误。

同时，也有很多 BOF 脆弱点可能不会用到危险函数。例如 CVE-2014-0782 脆弱点是在 rep movsd 处发生了错误，CVE-2012-1141 是在用 for

```
31D131EF
31D131EF loc_31D131EF:
31D131EF mov     eax, [ebx]
31D131F1 lea     esi, [eax+853Ch]
31D131F7 mov     edi, [esi]
31D131F9 mov     eax, [eax+80FCh]
31D131FF lea     edx, [eax+edi*8] ; Dst
31D13202 inc     edi
31D13203 push    8               ; Size
31D13205 mov     [esi], edi
31D13207 call    call_memcpy
31D1320C jmp     loc_31D1275F    ; jumptable 31D12721 default case
```

图 2 CVE-2014-1761 脆弱点出错位置

语句对 list→field 进行处理时发生了错误等。虽然产生 BOF 错误有多种方式，但是通过进一步分析，发现该类错误一般发生在循环中，例如上面提到的 rep movsd 指令，以及如图 3 所示的 xp sp3 中的 strcpy 函数。

```
.text:7C92250C   mov   dl, [ecx]
.text:7C92250E   inc   ecx
.text:7C92250F   test  dl, dl
.text:7C922511   jz    short loc_7C92257B
.text:7C922513   mov   [edi], dl
.text:7C922515   inc   edi
.text:7C922516   test  ecx, 3
.text:7C92251C   jz    short loc_7C922527
.text:7C92251E   jmp   short loc_7C92250C
```

图 3 strcpy 函数循环

对于无边界检查的 BOF 脆弱点的特征，可以归纳为如图 4 所示的模型：一是它发生在一个循环结构中；二是在该循环中存在读写内存操作；三是控制读写的程序指针能被程序输入污染，即能被用户控制。若满足这 3 个条件，则可以控制 Pointer 访问非法的地址，从而发生错误。

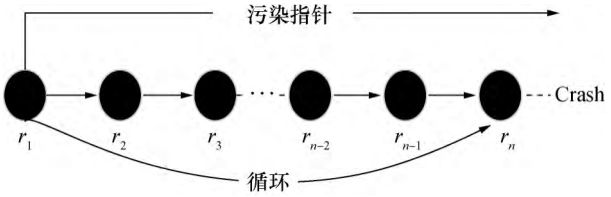

图 4 无边界检查 BOF 特征模型

2.2 有边界检查 BOF 特征

事实上，随着软件厂商对安全问题越来越重视，在对内存进行读写时，程序员对边界进行检查也变得越来越普遍。但是即使开发者加强了对程序安全的防范，实际应用中依然存在许多 BOF 问题。为深入分析有边界检查 BOF 特征，本文以 CVE-2011-0978 和 CVE-2014-0497 为例进行研究。

CVE-2011-0978 是 Office Excel 在解析 Axis Properties Record 属性时发生的错误。该脆弱点在使用数组索引时，虽然有判断条件，但是可被绕

过，导致溢出。具体出错代码如图 5 所示。

```
30177CED        mov   dword_3088D744, esi
30177CF3        cmp   edx, [eax+0Ch]
30177CF6        jge   loc_30324E3C
-------------------------------------
30177D1C        movzx  ecx, word ptr [eax]
-------------------------------------
30177D24        push   [ebp+Dst] ; Dst
30177D27        add    eax, 2
30177D2A        push   eax
30177D2B        CALL EXCEL.3000C293
```

图 5 CVE-2011-0978 出错代码区域

CVE-2014-0497 脆弱点具有类似的问题，也是由于数组越界产生的缓冲区溢出。

因此，从上面的分析可以看出，对于有边界检查 BOF 脆弱点，虽然进行了边界检查，但是由于检查方式存在缺陷，使得可以利用索引指针访问到数组外的数据，进而产生错误。该类 BOF 具体特征如图 6 所示：一是与有边界 BOF 类似，它发生在数组的循环访问操作中；二是它存在指针越界处理指令；三是越界处理指令的守护条件能被污染。

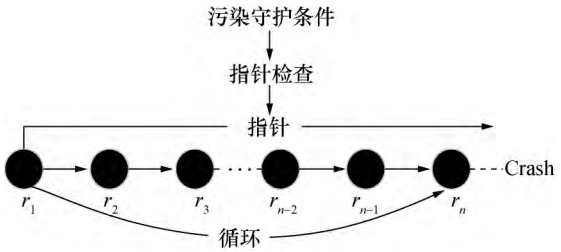

图 6 有边界检查 BOF 特征模型

2.3 可疑点收集

总结上述 2 种特征，本文通过静态分析，将程序中包含这 2 种模式的代码作为测试目标。但是通过这 2 种模式找到的代码规模仍然可能较大，例如在类似 while、for 循环中可能有许多其他功能的代码，因此有必要进一步缩减可疑测试代码。在本文分析的脆弱点中，BOF 出错的地方一般是利用数组指针进行访存操作。根据 Mitchell 等[14]的研究成果，数组 A 访问形式一共有 3 种：一是常量访问模式，即 A[const]＝…；二是步进访问模式，即 A[$f(i)$]＝…；三是非单调性的访问模式，即 A[$f(x)$]＝…。

对比 BOF 的特征，第一种模式一般不会产生问题，第二种数组访问模式在循环访问中比较容易产生溢出，对于第三种非单调性的访问模式，可能产生有边界检查的溢出。例如 CVE-2014-0497 就是 li32 函数根据需要多次访问 ByteArray 数组中的数据产生的溢出。为了进一步提高测试针对性，在已有分析的基础上，重点考虑覆盖可疑的指针，并做如下处理：假设指针 p 为一个循环中的指针，记为 $test_p$，待测试区域记为 TG($test_p$)，其中 TG 表示和测试指针 p 相关的测试区域(test group)，则 TG($test_p$)表示在循环执行中，所有能影响该指针值的指令；考虑到无边界 BOF 特征，必须考虑到影响 $test_p$ 的条件，例如 if(var＜10){＊p＋＋＝0;}指令，var 能直接控制在第一步需要测试的指针 p，记为 var＝guard(p)。因此，TG(guard(p))也是本文的测试目标。获得测试目标 TG(pointers)的算法具体如图 7 所示。

```
算法：获取测试目标
输入：二进制文件(bins)
输出：测试目标
1   Set TG[], loops[], pointers[], guards[]→Ø
2   disassemble the bins and construct its CFG
3   detectLoop()→loops[]
4   foreach loop in loops[]do
5     analysis the pointer and its alias in the loop
6     getPointer()→pointers[]
7   end foreach
8   foreach pointer in pointers[] do
9     getGuard()→guards[]
10  end foreach
11  foreach element in pointers[] and guards[] do
12    analysis the dataflow about element
13    getInstuctions()→TG[]
14  end foreach
15  return TG[]
```

图 7 获取测试目标算法

3 基于域收敛的路径遍历策略

一个二进制程序由许多基本块组成，敏感点包含在这些代码块或几个基本块的组成中。如果将一个代码块或多个代码块看成一个域，那么对目标点的遍历便等同于对域的遍历。

通过程序的控制流图，可以知道代码块之间的关系，结合程序动态获取的路径和反汇编信息，便能进入包含可疑点的域。此时，若将图 7 中收集的指令也比做一个域，对该域进行同样的处理后，则会命中目标指令。本文选择一个针对测试程序的样本集 T，从中取出测试用例 $t\in T$，假若它没有进入

可疑区域，则收集它执行的路径条件，并对路径条件进行操作分析，生成新的测试用例，使得下次执行路径能更接近目标域，然后按照这种方式逐步迭代，直到生成的测试用例能够覆盖目标，具体如图8所示。

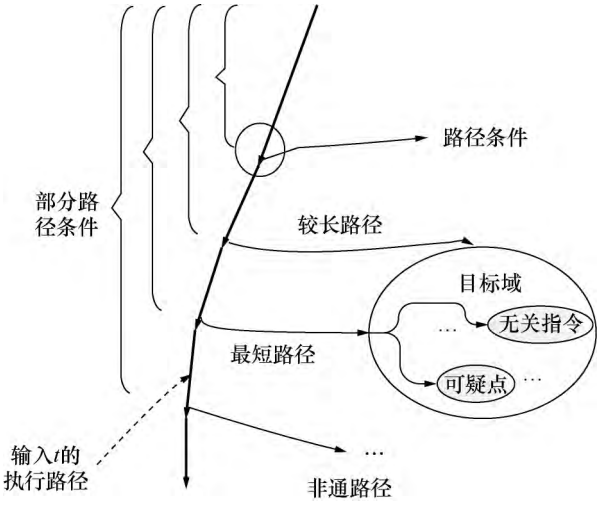

图8 基于域收敛的路径遍历

图8中间的粗线条表示一条没有命中的执行路径，本文把没有到达目标点的程序路径称为部分路径条件(partial path conditions，PPC)，然后利用PPC与目标域的距离来指导样本的优先构造顺序，从而去逼近或覆盖可疑指针。

假设一条执行路径没有进入程序指定的危险区域(danger)，它对应一个测试输入 t，t 的路径条件是 $\psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_m$。如果寻找从 t 逼近 danger 的输入，则可以从 t 的任何一个分支进行逼近。如果一个输入从分支的 k 处开始，那么它必须满足：

$$\mathrm{PPC} = \psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_{k-1} \wedge \psi_k.$$

也就是说，这个新的输入，满足原始输入 t 的前$(k-1)$个分支条件，但是不满足第 k 个分支条件。因而可以设计一个满足上式的新输入，让程序执行不同的输入，使得程序可能执行到目标处。另外，一般来说，若 PPC 与可疑目标点的距离较小，执行路径也较少，相应的测试效率也会更高。则本文取反的路径条件应该满足下面的条件：

$$\mathrm{PPC} \wedge \min |\, \mathrm{danger} - \mathrm{PPC} \,|.$$

如上所述，算法设计思路如下：首先，根据控制流图，计算路径上每个节点到特定目标点距离。在定义距离权重时，分为2种情况进行定义：基本块与基本块边的权重为1，辅助的边(如节点中的函数调用)的权重设为0；若命中包含目标指令的基本块，却没有命中需要的指令时，在基本块内指令与指令间的距离也定义为1。其次，利用 Dijkstra 算法[15]计算 PPC 与目标点的最短距离，指导条件取反，生成新的样本。例如，若程序 P 的一个路径分支为 b，对应的路径条件为 ψ，$\mathrm{dist}(\psi)$ 为 b 到特定目标的最短路径，若 $\mathrm{dist}(\psi)=0$，则表示已进入特定目标；若 $\mathrm{dist}(\psi)\neq 0$，则利用算法2，根据 PPC 的路径条件 $\varphi_i(\psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_i \wedge \psi_{i+1})$，选取合适的条件 ψ_i 进行取反，根据约束表达式，通过求解约束表达式构造新的测试用例，不断逼近测试目标，继而遍历所有的可疑目标点，具体如图9所示。

算法：基于域收敛的路径遍历算法
输入： P，t，unused(未使用的 PPC)，S(所有的 PPC)
输出： 程序 P 测试集 T_P

```
1  Set unused, S, T_P = ϕ
2  procedure Excute (P, T)
3    execute P with input T
4    let f = (ψ1 ∧ ψ2 ∧ ⋯ ∧ ψm) be the pt-condition
5    for all i from 1 to m do
6    φi =def= (ψ1 ∧ ψ2 ∧ ⋯ ∧ ψi ∧ ψi+1)
7    if dist(ψi+1) ≠ ∞ and dist(ψi+1) > 0 and φi ∉ S
8      S ∪ = φi , unused ∪ = φi
9    end if
10   end for
11 end procedure
12 procedure Produce(unused, danger)
13   While unused ≠ ∅ and not timeout do
14     Select φ ∈ unused with min-dist to danger
15     Remove φ from unused
16     Let ϕ = (ψ1 ∧ ψ2 ∧ ⋯ ∧ ψk−1 ∧ ψk)
17     Construct θ = (ψ1 ∧ ⋯ ∧ ψk−1 ∧ ¬ψk)
18     solve θ
19     if θ is satisfied then
20       let tθ be an input that satisfies θ
21      return tθ
22     else return null
23     end if
24   end while
25 end procedure
26 for all t ∈ T_P do
27   Execute(P, t)
28   t_new = Produce(unused, danger)
29   if t_new ≠ null then
30     T_P ∪ = t_new
31   end if
32   Remove t from T_P
33 end for
34 return T_P
```

图9 基于域收敛的路径遍历算法

4 实验与分析

为验证方法的有效性和正确性，本文构建了实验环境，选取真实的测试用例进行分析与验证。

4.1 实验环境

本文开发了基于 BOF 特征导向的 Fuzzing 原型工具 BOFuzz，分别在 Ubuntu 12.04 和 Windows XP SP3 下进行了实验，物理内存均为 4 GB。

4.2 可疑点探测能力测试

为了验证本文方法的有效性，对 GDI32.dll 等 5 个程序或模块进行了测试，具体测试结果如表 1 所示。

表 1 可疑点探测结果

程序	循环总数	可疑点	危险函数	增加比例/%
GDI32.dll	657	72	51	41
ssleay32.dll	112	45	30	50
freeType	2 568	412	249	65
FreeFloat FTP	146	21	12	75
CoolPlayer	1 036	160	56	185

以文[11]等为代表的方法，一般根据经验，将 strcpy 等危险函数作为可疑点来进行测试。从表 1 可以看出，本文的方法能更多地发现程序或模块中的可疑点，可疑点最少增加 41%，最多增加了 185%。同时，有些脆弱点出错的地方可能并非是由于调用了危险函数，例如 CVE-2010-2806。因此，本文可疑点识别方法，能更全面地收集可疑点，提高脆弱点的发现概率。

4.3 域收敛路径遍历算法性能测试

本文提出的基于域收敛的路径遍历算法，是为了缓解了路径空间爆炸问题。为测试本文算法的能力，本文针对 nginx，分别利用深度优先算法与域收敛路径遍历算法对其进行了测试。具体结果如图 10 所示。

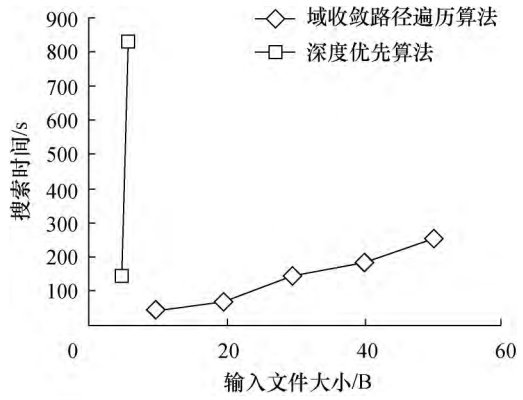

图 10 深度优先和域收敛路径遍历算法对比

从图 10 可以看出，对于深度优先算法，当输入文件为 5 B 时，其搜索时间大约需要 142 s；当输入文件增加到 6 B 时，搜索时间大约需要 830 s；当输入文件增加到 7 B 时，可以发现需要的时间已大于 1 h。从深度优先算法的搜索时间的趋势来看，很容易产生空间爆炸；而对于本文的域收敛路径遍历算法，当输入为 10 B 时，搜索时间需要 43 s；当输入增加到 50 B 时，搜索时间也仅需 257 s，可见时间增长并不明显，达到了缓解空间爆炸的目的。

4.4 脆弱点检测能力测试

为了验证本文方法对已发现脆弱点的检测能力，本文对 inspircd、OpenSSL 和 freeType 中的 4 个脆弱点进行了测试，表 2 是检测的具体结果，其中：D 表示可疑点个数，并且表中的可疑点只包括出错模块的数目；I 表示可疑点是否包含出错点；T 表示执行时间，单位为 s；S 表示是否验证成功。

表 2 脆弱点结果测试表

软件	脆弱点	D	I	T/s	S
inspircd	CVE-2012-1836	307	Y	42	Y
OpenSSL	CVE-2007-5135	45	Y	190	Y
	CVE-2014-0160	23	Y	>1 800	N
freeType	CVE-2010-2806	412	Y	401	Y

可以看出，对于上述 3 款软件，BOFuzz 能在较短的时间内检测出其中的 3 个脆弱点，说明本文方法在实际的脆弱点检测上具备较好的能力。同时，对于验证成功的 3 个脆弱点，其中 CVE-2010-2806 出错的地方是 t42_parse_sfnts()函数，CVE-2007-5135 中出错的地方是 SSL_get_shared_ciphers()函数，它们都不是明显的危险函数，如果利用文[11]中覆盖危险函数的方法将很可能无法发现该脆弱点。但是，BOFuzz 无法成功验证 CVE-2014-0160，尽管该脆弱点也是由于调用 memcpy 危险函数产生的错误。通过分析可以发现，该脆弱点虽然发生了溢出，但是却没有产生异常，只是发生了信息泄露，因而无法正确检测，这也正是本文方法的一个不足之处。

5 结 论

为了克服模糊测试的盲目性，缓解符号执行方法面临的空间爆炸问题，本文提出一种基于脆弱点特征导向的软件安全测试方法，该方法精确分析并识别了缓冲区溢出点，并以此为指导，利用域收敛路径遍历策略，有效提高了测试的针对性，缓解了

路径空间爆炸问题，对测试效率的提高有着显著效果。然而，本文仍然有许多需要解决的问题，例如在分析缓冲区溢出特点时，如何更有效地识别循环、数组指针等重要信息；以及如何发掘不会触发异常的缓冲区溢出等，还需在后续的工作中进一步研究。

参考文献 (References)

[1] 李红辉，齐佳，刘峰，等. 模糊测试技术研究[J]. 中国科学:信息科学，2014，44(10)：1305-1322.
LI Honghui，QI Jia，LIU Feng，et al. The research progress of fuzz testing technology [J]. Science China：*Information Sciences*，2014，**44**(10)：1305-1322.(in Chinese)

[2] 李舟军，张俊贤，廖湘科，等. 软件安全漏洞检测技术[J]. 计算机学报，2015，**4**：717-732.
LI Zhoujun，ZHANG Junxian，LIAO Xiangke，et al. Survey of software vulnerability detection techniques [J]. *Chinese Journal of Computers*，2015，**4**：717-732.(in Chinese)

[3] 杨丁宁，肖晖，张玉清. 基于 Fuzzing 的 ActiveX 控件漏洞挖掘技术研究[J]. 计算机研究与发展，2012，**49**(7)：1525-1532.
YANG Dingning，XIAO Hui，ZHANG Yuqing. Vulnerability detection in ActiveX controls based on fuzzing technology [J]. *Journal of Computer Research and Development*，2012，**49**(7)：1525-1532.(in Chinese)

[4] 李伟明，张爱芳，刘建财，等. 网络协议的自动化模糊测试漏洞挖掘方法[J]. 计算机学报，2011，**2**：242-255.
LI Weiming，ZHANG Aifang，LIU Jiancai，et al. An automatic network protocol fuzz testing and vulnerability discover method [J]. *Chinese Journal of Computers*，2011，**2**：242-255.(in Chinese)

[5] 欧阳永基，魏强，王清贤，等. 基于异常分布导向的智能 Fuzzing 方法[J]. 电子与信息学报，2015，**37**(1)：143-149.
OUYANG Yongji，WEI Qiang，WANG Qingxian，et al. Intelligent fuzzing based on exception distribution steering [J]. *Journal of Electronics and Information Technology*，2015，**37**(1)：143-149.(in Chinese)

[6] 马金鑫，张涛，李舟军，等. Fuzzing 过程中的若干优化方法[J]. 清华大学学报(自然科学版). 2016，**56**(5)：478-483.
MA Jinxin，ZHANG Tao，LI Zhoujun，et al. Improved fuzzy analysis methods [J]. *Journal of Tsinghua University (Science and Technology)* 2016，**56**(5)：478-483.(in Chinese)

[7] Cadar C，Dunbar D，Engler D R. KLEE：Unassisted and automatic generation of high-coverage tests for complex systems programs [C]// Proceedings of OSDI'08. San Diego，CA，USA：USENIX Association，2008：209-224.

[8] Sen K，Agha G. CUTE and jCUTE：Concolic unit testing and explicit path model-checking tools [C]// Proceedings of Computer Aided Verification. Berlin Heidelberg，Germany：Springer，2006：419-423.

[9] Chipounov V，Kuznetsov V，Candea G. S2E：A platform for in-vivo multi-path analysis of software systems [J]. *ACM SIGARCH Computer Architecture News*，2011，**39**(1)：265-278.

[10] Godefroid P，Levin M Y，Molnar D. SAGE：Whitebox fuzzing for security testing [J]. *Queue*，2012，**10**(1)：20.

[11] 崔宝江，梁晓兵，王禹，等. 基于回溯与引导的关键代码区域覆盖的二进制程序测试技术研究[J]. 电子与信息学报，2012，**34**(1)：108-114.
CUI Baojiang，LIANG Xiaobing，WANG Yu，et al. The study of binary program test techniques based on backtracking and leading for covering key code area [J]. *Journal of Electronics & Information Technology*，2012，**34**(1)：108-114.(in Chinese)

[12] Haller I，Slowinska A，Neugschwandtner M，et al. Dowsing for overflows：A guided fuzzer to find buffer boundary violations [C]// Proceedings of 22nd USENIX Security Symposium. Washington DC，USA：USENIX Association，2013：49-64.

[13] Patrice G. Compositional dynamic test generation [C]// Proceedings of ACM Sigplan Notices. New York，NY，USA：ACM Press，2007：47-54.

[14] Mitchell N，Carter L，Ferrante J. A modal model of memory [C]// Proceedings of International Conference on Computational Science. Berlin Heidelberg，Germany：Springer，2001：81-96.

[15] Edsger W D. A Discipline of Programming [M]. Upper Saddle River：Prentice Hall，1997.