doi:10.3969/j.issn.1000-436x.2013.z2.013

PyFuzzer：自动化高效内存模糊测试方法

李伟明，于俊清，艾少波
（华中科技大学 网络与计算中心，湖北 武汉 430074）

摘 要：针对传统模糊测试（fuzz testing）耗时、无法绕过有效性验证等缺陷，提出了基于快速内存模糊测试，综合运用静态分析和动态跟踪技术的测试工具—PyFuzzer。整个过程高度自动化，通过 WarFTPD、Serv-U 等程序进行测试，并和 4n FTP Fuzzer 进行对比，结果表明 PyFuzzer 能有效地发掘二进制程序中的各种漏洞，极大地提高了模糊测试的效率。

关键词：模糊测试；静态分析；动态跟踪；漏洞挖掘
中图分类号：TP311.134.3 **文献标识码**：A **文章编号**：1000-436X(2013)Z2-0064-05

PyFuzzer：automatic in-memory fuzz testing method

LI Wei-ming, YU Jun-qing, AI Shao-bo
(Network and Computation Center, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: Fuzz Testing is an effective method to mine all kinds of vulnerabilities. But the main drawbacks to current fuzz testing tools are: firstly, it produces high volume testing data and it's extraordinary time consumption; secondly, if the accessing needs authentication, the greatest part of test data will be abandoned. PyFuzzer, a novel automatic in-memory fuzz testing tool combining static analysis, dynamic analysis and in-memory fuzz testing, was presented. The tool is highly automatic and effective. Compared with 4n FTP Fuzzer in testing WarFTPD and Serv-U, PyFuzzer can discover all vulnerabilities and improve test efficiency greatly.

Key words: fuzz testing; static analysis; dynamic tracking; vulnerabilities excavate

1 引言

中国国家信息安全漏洞库[1]在 2013 年仅 5 月就发布了 647 个各种安全漏洞。如果被攻击者利用，将会产生巨大危害。因此，本文提出了一种自动化识别危险函数并进行内存模糊测试（in-memory fuzz testing）的方法，能够有效减少人工分析的干预，提高模糊测试的效率，发掘程序中缓冲区溢出、拒绝服务和格式化字符串等漏洞。

2 介绍

在安全漏洞挖掘领域，fuzz testing 受到了学者较深入的研究。它是一种不断向程序输入畸形数据来使执行程序出现问题的动态测试技术。最早由 Miller[2]利用 fuzz testing 测试 Unix 程序，Forrester[3]采用类似方法对 Windows GUI 程序进行 fuzz testing，发现了程序中的大量漏洞。2002 年，文献[4]提出了利用文件格式知识和协议知识构造测试用例来进行 fuzz testing，提高了测试用例的有效性，但难以实现高度自动化，效率较低。文献[5]提出了协议自动化分析来获取知识，但当前协议逆向分析效果不理想，自动化漏洞挖掘效果还不如人工分析方法。另外普通的 fuzz testing 无法突破应用程序中的验证或检查[6]，无法进行更为深入的测试。

In-memory fuzz testing 的提出[7]可以解决这个难题。In-memory fuzz testing 的思想最初由 Greg Hoglund 在 2003 年 Blackhat USA 安全会议上提出。Corelan 团队利用这种方法发掘出了 IE 浏览器中的多

收稿日期：2013-09-11
基金项目：国家自然科学基金资助项目(61370230)
Foundation Item：The National Natural Science Foundation of China(61370230)

个漏洞[8,9]。

在本文中提出了一种基于内存模糊测试系统 PyFuzzer，其综合利用了静态特征分析[10]和动态污点跟踪的技术。图 1 描述了 PyFuzzer 的核心流程，分为三部分：静态特征分析、动态跟踪和 In-memory fuzz testing。

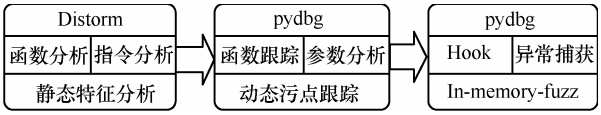


图 1　PyFuzzer 组成结构

3　静态特征分析

在 fuzz testing 测试中，函数被作为基本测试单位。针对 Windows 平台下的可执行程序，可能导致漏洞的危险函数可以分为以下几类：1) 来自库的危险函数。例如标准 C 库中的 strcpy 等；2) 用户自定义的缺乏边界检查、存在潜在危险的函数。通过对存在漏洞的函数反汇编分析，发现普遍存在缺少边界检查的情况。另外，在实验中发现，缓冲区连续数据拷贝操作常用串操作指令 rep movsd/w/b 来实现。在用动态调试器 OllyDbg 调试验证已知漏洞时，发现 rep movsd 指令原操作数内存地址被覆盖，导致内存不可读，引发访问异常。

综上可知，把易存在漏洞的危险函数可以划分为如下几类：来自库函数中的易受攻击函数、循环字符拷贝指令序列 JXX 结构、串拷贝指令 rep mov 和处理输入数据函数。最后一种危险函数会在动态跟踪中进行说明。

在对二进制程序静态解析和发掘危险函数时利用了 Distorm 反汇编引擎；在可执行程序的函数解析中主要使用了基于 call\ret、jmp 指令的函数调用方法，并利用了 PE 文件中的导入表来得到二进制程序的所有函数列表。得到所有的函数后，并不是直接对函数进行 fuzz testing 测试，而是利用静态分析方法来为模糊测试器找出潜在的高危函数，节省时间并提高效率。

本文使用测试优先级的方法对危险函数进行优先级的评估。对常见危险函数利用黑名单的方法进行评估。C 库函数 malloc、alloc、strcpy、strncpy 等给予较高的优先级；对处理用户输入数据的函数，因为其被用户数据控制，存在漏洞概率高，同时在黑名单内的函数给予高优先级；对包含循环拷贝指令序列和串拷贝的函数也是同样处理；最后再给剩余的函数低优先级，如表 1 所示。

表 1　　函数测试优先级

函数	黑名单	污点数据	优先级
strcpy	Y	Y	高
循环拷贝指令	Y	Y	高
fgets	Y	N	中
foo	N	N	低

为了匹配 JXX 型指令序列结构，采用了 Needleman Wunsch 比对算法。因为没有规范化的指令序列直接和典型结构匹配，效果很不理想。所以在匹配之前，需要对指令序列进行规范化处理，然后再计算 2 个序列的 Needleman 距离。把指令操作数用符号 *Operator* 表示，共分为 6 类。为了方便字符串比对，把每一种类别用一个小写字母表示，字符规范化处理后如表 2 所示。

表 2　　操作数类型与字符对照表

操作数类型	字符符号
OPERAND_NONE = ""	*x*
OPERAND_IMMEDIATE = "Immediate"	*i*
OPERAND_REGISTER = "Register"	*r*
OPERAND_ABSOLUTE_ADDRESS	*a*
OPERAND_MEMORY = "AbsoluteMemory"	*m*
OPERAND_FAR_MEMORY = "FarMemory"	*f*

即

$$Operator=\{x, i, r, a, m, f\}$$

在 x86 中，指令多达几百条，但考虑到实际测试目标中使用的指令类型非常有限，所以可以简化处理。在对指令操作符的规范化处理中，把操作符用能表示指令基本意义的单个大写字母表示。指令操作符用 Operation 表示。例如 mov 记作 M，inc 记作 I，test 记作 T，jo、jno、jb、jnb、je、jne、ja、js、jns、jp、jnp、jl 记作 J。注意，push 和 pop 指令都会被记作 P，这是允许的，因为连续拷贝典型结构中没有这两条指令，在使用 Needleman/Wunsch 算法时不会对结果造成影响。

Operation={*M*,*I*,*T*,*J*,*P*,*O*,*A*,*X*,…};这样就可以构造指令串 *string*=$c_1c_2[c_3]$，c_1 属于 *Operation*，c_2、c_3 属于 *Operator*。如果是单操作数指令，则没有 c_3。图 2 的典型结构就可以记作 *MrmMmrIrTrrJi*。设 *Dist* 为 2 个串的 Needleman 距离，则

$$Dist=\min(Needleman(S,r))$$

其中，$S=\{P_1,P_2,P_3,\cdots,P_n\}$为模式串的集合，$P_i(i=1,2,3,\cdots,n)$为模式串，$P=string_1\ string_2\ string_3\cdots string_m$，$m$ 为自然数；在目标程序反汇编代码中得到一系列的循环结构，在对其规范化处理后得到循环结构集合 R。通过 Needleman 算法计算出循环指令结构与模式串的 Needleman 距离，其值越小，相似度越高。通过对大量循环结构观察分析，选取 15 作为阈值，当 Needleman 值小于 15 时，则认为是循环字符拷贝指令序列。

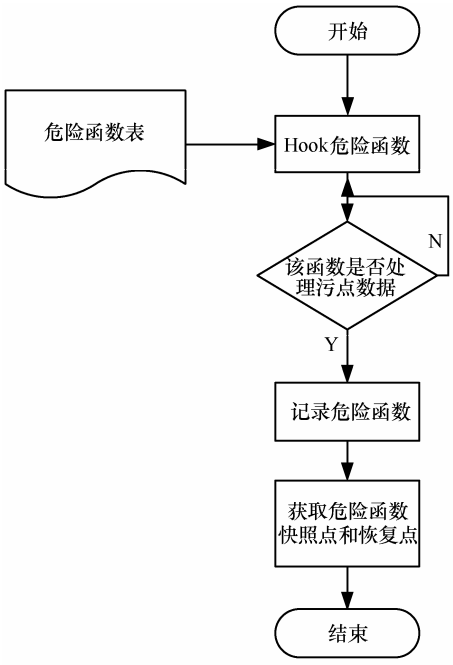


图 2　动态跟踪流程

4　基于动态跟踪的危险函数识别

在静态分析后，会得到危险函数记录文件。利用 Pydbg 函数库对所有危险函数进行 Hook。当发现被 Hook 的函数对输入数据进行处理，就触发了回调函数记录该函数的入口地址，返回地址及函数参数信息。基于动态跟踪的危险函数识别主要框架如图 2 所示。

4.1　危险函数跟踪

由于用户的输入是不可信的，在程序执行过程中，输入数据流向的所有函数都会导致该函数被污染，即该函数可能存在潜在威胁。Hook 函数和处理输入数据部分是指，在程序的执行流中对所有危险函数进行检查其是否要处理输入数据，如果发现函数被污染，则记录函数信息；反之则继续执行。可以根据危险函数记录对所有危险函数起始地址设置断点(INT 3)，这样在程序执行到危险函数时就会触发异常，在异常处理中设置回调函数，这样在回调函数中可以完成对危险函数的处理。

4.2　危险函数参数分析

由于函数调用通常要遵守其调用约定。C 语言的调用约定为 cdecl，微软为_stdcall，其规则都是首先通过 push 指令把参数压栈，然后通过 call 指令来调用函数。在回调函数中通过解析 push 指令压栈的数据就可以分析出危险函数的参数。

在获得函数参数后，对参数所指向的数据与输入数据比对。如果匹配成功，则说明该函数会处理输入数据，记录相关参数和该函数的起始地址，返回地址和参数值；如果匹配失败，则说明该函数不处理输入，继续处理下个危险函数。在目标程序执行完后，会得到真正需要模糊测试的危险函数信息记录，其本质是一系列顺序的函数调用过程。

5　内存模糊测试

5.1　内存模糊测试原理

模糊测试通常作用于程序外部，根据一定策略构造畸形数据并发送给目标程序。而内存模糊测试几乎全部作用于程序内部，只需在开始进行测试时对程序发送一次数据，内存模糊测试方法就可以在目标进程内自动生成测试用例的畸形数据，其循环测试过程也不需外界干预，这样就可以减少程序 I/O，提高效率。本文采用基于快照恢复的内存模糊测试方法，其主要步骤分为以下三部分。

1) 确定模糊测试的快照点和恢复点；快照点是指程序运行时的某刻对进程进行快照，保存进程运行所需要的全部信息，包括进程执行上下文、系统环境变量、全局变量等。恢复点是指程序运行到某刻（通常是函数的返回地址）把进程上下文恢复到快照点的位置，使进程能从快照点处开始执行。

2) 定位输入数据位置，生成测试用例；在动态跟踪时对危险函数解析后，可以获得函数的参数信息。函数的参数值即是输入数据的起始地址。在进程空间内分配缓存并生成测试用例的畸形数据填

充缓存，然后修改测试目标的参数，使其指向分配的缓存。这样就可以在进程内部完成测试用例数据的生成，不用从外部发送，避免了 I/O 且提高了测试效率。

3) 循环进行内存模糊测试；为了让测试不停地循环执行，采用了基于快照恢复的内存模糊测试方法。在快照点处保存进程快照，变异测试函数所指向的内存数据，等到程序运行到恢复点处时恢复进程快照，就可以回到快照点处进行下一轮的测试。

5.2 In-memory fuzz testing 模块的实现

在这一部分中，使用 Pydbg 库，利用其提供的调试类和调试函数实现了 In-memory fuzz testing 模块。该模块由 3 部分构成：Fuzzlib、回调模块、注册模块。其中回调模块分为软件断点回调和访问异常回调。

1) Fuzzlib。Fuzzlib 用于构造畸形测试数据。例如，由“\x0a”，“\x0d”，“,”，“.”，“:”，“;”，“&”，“%”,“$”,“\x20”,“\x00”,“#”,“(“,”)”，“{“,”}”，“< “,” >”，“\ “”，“”，“\\”，“|”，“@”，“*”，“-”，“\t”等字符构成的字符串或二进制串。

2) 软件断点回调。它用来处理来自前面静态特征分析和动态分析得到的危险函数。在目标测试中，根据快照点和恢复点处设置软件断点（INT 3），当程序运行到该处时就会触发断点回调。断点回调处理算法如下：

① 如果断点是快照点，若没有获得进程快照，则拍下进程快照；

② 在目标进程空间分配内存空间，利用 Fuzzlib 生成恶意数据，写入内存；

③ 记录生成恶意数据；

④ 修改函数参数，使其指向②中的内存空间；

⑤ 重新设置恢复点后跳至⑦；

⑥ 如果是恢复点，则恢复进程快照，并重新设置恢复点；

⑦ 进入继续调试状态。

3) 访问异常回调。当出现异常时，记录当前进程寄存器上下文，反汇编代码。

4) 注册回调。在对程序调试前，注册前面提到的软件断点回调和访问异常回调。

6 PyFuzzer 测试与分析

利用 PyFuzzer，主要对 WarFTPD、Serv-U 等网络程序进行测试。实验的测试环境如下：Windows 7 SP1，Python 2.7，Distorm 3.0，pydbg，Serv-U build 4.0.0.4，WarFTPD 1.65，4n FTP Fuzzer 。

PyFuzzer 对 WarFTPD 和 Serv-U 进行了测试，首先通过静态分析，实验结果如表 3 所示。

表 3　　静态分析结果

测试程序	函数数目	危险函数数目	JXX 循环拷贝结构	Rep mov
WarFTPD 1.65	1 987	362	6	432
Serv-U build 4.0.0.4	2 392	580	13	719

其次本文采用了网络流行 FTP 测试工具 4n FTP Fuzzer[15]与 PyFuzzer 测试对比。在对 WarFTPD 进行测试时，由于 4n FTP Fuzzer 不稳定，导致异常退出。所以选择对 Serv-U 服务器进行测试，测试的结果如表 4 所示，都能发现存在的漏洞。

表 4　　对比测试结果

程序/(工具)	测试结果	漏洞类型
WarFTPD1.65 (PyFuzzer)	CWD, CDUP, DELE,NLST, LIST	DoS
	USER	Buffer overflow
Serv-U build 4.0.0.4 (PyFuzzer)	SITE CHMOD,MDTM, LIST	Buffer overflow
	XCRC, STOU, DSIZ	DoS
Serv-U build 4.0.0.4 (4n FTP Fuzzer)	SITE CHMOD,MDTM, SMNT	Buffer overflow
	STOU	DoS

在测试效率上，PyFuzzer 测试效率为 113 次/s，而 4n FTP Fuzzer 的测试效率只有 48 次/s。PyFuzzer 在单位时间内测试的次数多于 4n FTP Fuzzer。测试表明在测试效率上，PyFuzzer 较从外部输入的测试工具有极大的提高。

7 结束语

本文提出了综合利用静态分析和动态分析的 In-memory fuzz testing 测试方法。设计并实现了该方法的原型 Pyfuzzer。Pyfuzzer 的动态跟踪模块可以监视跟踪用户输入。In-memory fuzz testing 的过程中能够利用动态跟踪的结果极大地提高测试效果，减少对无价值目标的测试工作。In-memory fuzz testing 技术可以绕过软件中的校验机制，既方便了测试，也提高了测试速度。在对 Serv-U、WarFTPD 测试实验中，发现并验证了大量存在的漏洞，实验结果表明

了 Pyfuzzer 方法的有效性。

参考文献：

[1] 中国国家漏洞库 [EB/OL].http://www.cnnvd.org.cn/vulnerability/, 2013.

[2] BARTON M, LOUIS F, BRYAN S. An empirical study of there liability of UNIX utilities[J]. Communications of the Association for Computing Machinery, 1990, 33(12):32-44.

[3] JUSTIN F, BARTON M. An empirical study of the robustness of Windows NT applications using random testing[A]. Proceedings of the 4th USENIX Windows System Symposium[C]. Berkeley: USENIX Association, 2000. 1-6.

[4] AITEL D. The Advantage of Block-Based Protocol Analysis for Security Testing[R]. Miami Beach: Immunity Inc, 2002.

[5] 何永君. 基于动态二进制分析的网络协议逆向解析[D]. 郑州: 解放军信息工程大学, 2010.
HE Y J. Network Protocol Reverse Parsing Based on Dynamic Binary Analysis[D]. Zhengzhou: PLA Information Engineering University, 2010.

[6] WANG T, WEI T, GU G, *et al*. TaintScope: a checksum-aware directed fuzz testing tool for automatic software vulnerability detection[A]. Proceedings of the IEEE Symposium on Security and Privacy[C]. Oakland, CA, 2010.497-512.

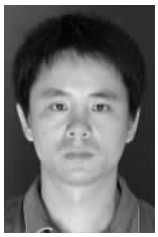
[7] SUTTON M, GREENE A, AMINI P. Fuzz testing: Brute Force Vulnerability Discovery[D]. Addison-Wesley Professional, 2007.13-15.

[8] IOZZO V. Knowledge fuzz testing [A]. Proceedings of the Black Hat[C]. 2010.

[9] 吴志勇, 夏建军, 孙乐昌. 多维 Fuzz testing 技术综述[J]. 计算机应用研究, 2010, 27(8):2810-2872.
WU Z Y, XIA J J, SUN L C. Survey of multi-dimensional fuzz testing technology[J]. Application Research of Computers, 2010, 27(8):2810-2872.

[10] EVANS D, LAROCHELLE D. Improving security using extensible lightweight static analysis[J]. IEEE Software, 2002,19(1):42-50.

作者简介：



李伟明（1975-），男，湖南株洲人，华中科技大学副教授，主要研究方向为网络安全。

于俊清（1975-），男，内蒙古赤峰人，华中科技大学教授、博士生导师，主要研究方向为数字媒体处理与检索、多核计算与流编译。

艾少波（1988-），男，湖北荆州人，华中科技大学硕士生，主要研究方向为网络安全。