基于模型的Fuzzing测试脚本自动化生成

侯 莹 洪 征 潘 璠 吴礼发
(解放军理工大学指挥自动化学院 南京 210007)

摘 要 针对基于知识的Fuzzing测试技术存在脚本编写工作量大的问题,提出一种基于模型的Fuzzing测试脚本自动生成方法。方法首先以高阶属性文法形式化地描述数据模型,获取统一的、与测试环境无关的数据格式描述;然后依据文法模型,将样本解析为带格式知识的文法分析树;最后建立文法分析树与测试逻辑的关联关系,实现自动化的测试脚本生成。实验结果表明,所提出的方法能够自动生成有效的测试脚本,并发现软件中潜在的安全漏洞。

关键词 高阶属性文法,模糊测试,文法分析树,漏洞挖掘
中图法分类号 TP393.08 **文献标识码** A

Model Based Automatic Fuzzing Script Generation

HOU Ying HONG Zheng PAN Fan WU Li-fa
(Institute of Command Automation, PLA University of Science & Technology, Nanjing 210007, China)

Abstract Knowledge based fuzzing techniques generally have some shortcomings of heavy workload in writing scripts. A model based automatic fuzzing script generation method was proposed. Firstly data format is represented by the higher order attribute grammars, and uniform data representation which is irrelevant to the test environments can be obtained. Secondly, the grammar model is used to parse the sample data and build the grammar parsing tree. Lastly, the relationship between the parsing tree and the test logic is built, which can be used to generate test script automatically. Experimental results indicate that the method can generate effective test scripts automatically to discover potential vulnerabilities in software.

Keywords High-order attribute grammar, Fuzzing test, Grammar parsing tree, Vulnerability mining

1 引言

Fuzzing测试又称为模糊测试,它通过提供非预期的输入并监视异常结果来发现软件的故障,广泛应用于软件安全性测试和漏洞挖掘领域。基于知识的Fuzzing测试[1]能够显著地提高测试用例的有效性,是当前研究的主流方向。其基本思想是以文件或协议格式规范为指导,选择样本中的字段进行变异,从而实现对目标程序的高效测试。然而,现有基于知识的Fuzzing测试平台主要依赖人工对样本的分析实现测试脚本生成,测试过程费时费力。同时,基于样本变异的方法对样本数据有很强的依赖性,单个样本难以完全覆盖数据格式的所有字段。采用大规模的样本数据集进行测试可以克服这一缺陷[2],但由于缺乏对复杂数据格式的统一描述能力,现有方法需要根据样本的不同格式编写相应的测试脚本,进一步增加了测试人员的负担。

为了克服上述缺陷,本文提出一种基于模型的模糊测试脚本自动生成方法。方法采用高阶属性文法形式化地描述数据格式,并通过对样本的解析得到文法分析树,最终实现从抽象模型到具体测试脚本的自动转化,从而避免了繁冗的脚本编写工作。

2 基于高阶属性文法的数据描述模型

2.1 数据格式分析

文件和协议一般都具有严格的格式规范,为了方便描述,本文将文件和协议统称为数据。从语言学的角度看,数据格式就是对数据构成单元词法、语法和语义的约定。词法为数据单元的符号特征;语法为数据单元的组织规则;语义为数据单元的具体解释。

通过对大量的文件和协议格式进行分析,我们发现数据格式中还存在如下两类依赖关系:

(1)语义与词法相关。以Emule协议为例,如图1中Emule协议的服务器消息报文和服务器列表报文中"消息长度字段"的语义与后续字段所占字节数(词法特征)相关。

(2)语义与语法相关。对比图1中服务器消息报文和服务器列表报文,前3个字段相同,后续字段的构造方式(语法特征)由"操作码"字段的语义决定:当"操作码"为0x38时,后续字段结构如图1(a)所示;当"操作码"为0x32时,后续字段结构如图1(b)所示;此外服务器列表报文中,服务器数字段

到稿日期:2012-05-21 返修日期:2012-08-17 本文受江苏省自然科学基金项目(BK2011115),军用网络技术实验室创新开放基金项目资助。
侯 莹(1988—),女,硕士生,主要研究方向为模糊测试、漏洞挖掘,E-mail:yinghou26@163.com;洪 征(1979—),男,博士,副教授,主要研究方向为信息与网络安全;潘 璠(1987—),男,博士生,主要研究方向为协议逆向分析、漏洞挖掘;吴礼发(1968—),男,博士,教授,主要研究方向为信息与网络安全。

的语义与后续服务器信息结构个数(语法特征)相关。

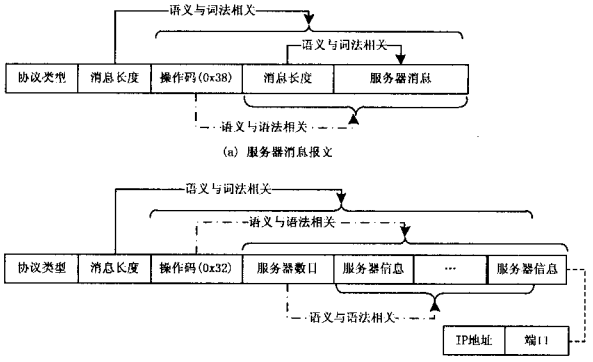

图 1 Emule 协议格式分析

SPIKE[3]作为最早的 Fuzzing 测试框架,提出基于块的表示技术来将协议描述为块序列的结构,并采用块嵌套的方法表示报文字段之间的包含关系和并列关系,但 SPIKE 字段间关系的描述仅限于长度约束,如图 1 中消息长度字段和后续结构体之间的约束关系。Peach[5]、Sulley[6]等沿用 SPIKE 的思想,通过添加属性标签,扩大了字段间依赖关系的描述,如图 1 中的校验和约束等。文献[7]提出了上下文无关文法形式的文件规范定义,通过推导式实现了对文件结构的描述,但其无法表示字段的语义。上述描述方法都难以表示图 1 中语义与语法相关的约束关系,无法对具有多种格式类型的复杂协议进行统一描述,在编写测试脚本时工作量巨大。

2.2 模型定义

属性文法 AG(Attributed Grammar)[8]于 1968 年被 Knuth 提出,因其语义表达能力强大并且具有可推理性,被广泛应用于描述和定义知识。高阶属性文法 HAG(High-order Attributed Grammar)[9]在属性文法的基础上,消除了属性和结构之间的界限,提高了对上下文相关数据的表达能力。由于文件和协议属于典型的上下文相关数据,本文采用高阶属性文法描述数据格式,将数据的组织结构也作为一种属性参与属性计算。以文献[9]为基础,对数据描述模型的形式化定义如下。

定义 1(数据格式描述模型) 数据格式描述模型为一个七元组 $DM=\{T,N,P,Z,A,R,NA\}$,其中:

1)T 为终结符集合,描述数据结构中不可分割的数据域。

2)N 为非终结符集合,描述可分解的结构体。

3)P 为结构体生成式的集合,生成式 $p(p\in P)$具有如下形式:

$p:x_0\rightarrow x_1\cdots x_{n-1}(x_0\in N,x_k\in T\cup N,1\leqslant k<n)$

4)Z 为推导数据格式生成的起始非终结符,$Z\in N$。

5)A 为属性符号集合,对任意 $x\in T\cup N$,都有一属性符号集,记作 $A(x)$且 $A=A\bigcup_{x\in T\cup N}A(x)$。符号 x_i 的 a 属性记为 $x_i.a$。

6)R 为属性规则集合,$R=\bigcup_{p\in P}R(p)$。$R(p)$用于描述生成式 p 中数据单元之间的约束关系。

7)NA 为非终结属性集合,$NA=\bigcup_{p\in P}NTA(p)$。$\forall p:x_0\rightarrow x_1\cdots x_{n-1}\in P$,非终结属性 NTA 可定义为:

$NTA(p)=\{x_j\mid x_j=f(\cdots x_i.a\cdots)\in R(p),0<i<j<n,x_j\in N\}$

式中,x_i,x_j 在 p 中出现,a 为 x_i 的属性,f 为属性计算函数。

非终结属性并非真正意义上的属性,它是指参与属性计算的非终结符,便于描述属性与数据组织结构之间的约束关系。若 $x\in NA$,记其为 $\overline{x}$。图 1 所示的 Emule 协议格式中,数据字段的组织结构由操作码字段的值决定,在属性规则中可描述为$\overline{DATA}=f(opcode.val)$,非终结符 $DATA$ 作为属性参与了计算,因此 $DATA\in NA$。

2.3 Emule 协议描述模型

根据 Emule 协议格式规范[10],构建的 Emule 协议的描述模型如图 2 所示,其中大写字符表示结构体(非终结符),小写字符表示数据域(终结符)。

DM={
T={Emule 协议报文的所有构成字段};
N={Emule 协议报文的字段组合,即结构体};
Z=EMULE;
P={
p_1:EMULE→protocol+size+opcode+$\overline{\text{DATA}}$
R(p_1):protocol. len=1;protocol. val=0xE3;opcode. typ=binary;
size. len=4;size. type=int;size. val=DATA. len+opcode.
len;protype. len=1;opcode. val={0x01,0x15,0x32,0x38,
0x40…… }
$\overline{\text{DATA}}$= Switch(opcode. val)
{ case 0x01:$\overline{\text{DATA}}$=LOGIN;
case 0x38:$\overline{\text{DATA}}$=SERVERMSG;
case 0x32:$\overline{\text{DATA}}$=SERVERLIST;
…}
p_2:SERVERMSG→msgsize+ MSG
R(p_2):msgsize. len=2;msgsize. type=int;msgsize. val=MSG.
len;
p_3 MSG→message+delim+MSG,1
R(p_3):message. type=string;
p_4:SERVERLIST→servercount+$\overline{\text{SERVERENTRY}}$
R(p_4):$\overline{\text{SERVERENTRY}}$=Count(ervercount. val);
servercount. typ=int
p_6:SERVERENTRY→ip+port
R(p_6):ip. len=4;port. len=2;ip. typ=String;port. typ=int;
}

图 2 Emule 协议的文法模型

在图 2 中,文法模型从起始符 Emule 开始推导,生成式用于描述 Emule 协议的物理组织结构。例如生成式 p_1 表示 Emule 协议报文由协议标签(protocol)、报文长度(size)、操作码(opcode)和报文数据(DATA)组成,“+”表示两个数据单元的连接操作。属性规则描述了生成式中文法元素的属性特性和依赖关系,综合 size 字段的属性规则可以看出该字段的长度(len)为 4 字节,类型(typ)为数值型,值(val)属性是报文数据(DATA)所包含的字节数。非终结属性的属性规则说明了服务器信息结构体与服务器数目字段(servercount)之间的计数关系;非终结属性的属性规则说明了 DATA 结构体与报文操作码字段(opcode)之间的依赖关系,将 Emule 协议不同类型的报文都纳入了模型表示;实现了同一规范下多种数据格式的统一描述,解决了传统模型中描述冗余的问题。此外,生成式 p_3 以递归的方式描述了服务器消息数据块(MSG)是由单行或多行文本消息组合而成,也表明了该模型对文本数据和二进制数据的描述都具有实用性。由于篇幅限制,图 2 只给出了 Emule 协议的服务器消息报文和服务器列表报文的主要生成式及属性规则。

3 测试脚本的自动化生成

文法模型是对数据格式的抽象描述，测试脚本是对具体文件或协议报文格式的描述，要实现脚本的自动化生成，需要在抽象模型与具体模型之间进行转化。模型转换的基本方法如图 3 所示：以抽象模型和样本数据为输入，根据生成式和属性规则对样本数据进行解析，生成文法分析树作为中间表示模型，进而采用具体的脚本描述语言描述已解析的数据格式，生成测试脚本。

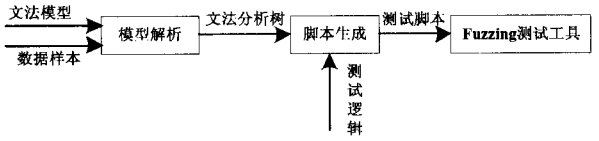

图 3 基于模型的自动化脚本生成流程图

3.1 文法分析树的构造

文法分析树是文法模型的实例化，是文法模型对样本数据进行解析后得出的中间数据模型。文法分析树的构造采用如下思路：以非终结符为中间节点，以终结符为叶节点，自顶向下构建文法分析树基本框架；根据文法元素的属性规则确定样本数据中域的边界，并将域取值作为节点属性加入到文法分析树中。具体的构造算法如下：

算法以文法模型 M、样本数据 *sample* 和起始非终结符 V 为输入，第 1～3 行为起始非终结符创建根节点；第 4 行在文法模型中搜索(Search)非终结符 V 的生成式；第 5～15 行遍历生成式的文法元素，识别匹配样本数据，其中分为 3 种情况：

1)若为终结符(第 6～8 行)，根据属性规则(M. R)提取(Match)该终结符在样本数据中的取值，创建节点(Node)加入分析树；

2)若为带非终结属性的非终结符(第 10～11 行)，则先根据属性规则和所依赖的节点值提取(NACalc)出该非终结符的值，再按照非终结符的处理方法构造分析树；

3)若为非终结符(第 12～15 行)，先为该非终结符创建节点加入树中，然后递归调用文法分析树构造算法(Gratree_Build)，生成以该非终结符为根节点的子树，最后用一个并操作将子树作为子节点加入到文法分析树中。

算法 1 文法分析树构造算法

输入：文法模型 M，单个数据样本 Sample，非终结符 V(初始状态为起始非终结符)；
输出：样本数据的文法分析树 G；

```
Gratree_Build (M,Sample,V) {
1. if(V==M. Z)
2.   fnode←Node(V);
3.   G. root←fnode;
4. p←Search(V);
5. for each v in p
6.     if(v∈T)
7.       v. val←Match(v,Sample,M. R);
8.       fnode. child←Node(v);
9.     else
10.      if(v∈NA)
11.        v←NACalc(v,T,M. R);
12.    fnode. child←Node(v);
13.    fnode←fnode. child;
14.    DG← Gratree_Build (M,Sample,v);
15.    G←G∪DG;
16. return G;
17. }
```

根据算法 1，以表 1 中的 Emule 协议的服务器消息报文样本为例，结合图 2 中 Emule 协议的文法模型，构造的文法分析树如图 4 所示。

表 1 服务器消息报文样本

域的名称	域值
protocol	0xE3
size	50
opcode	0x38
msgsize	47
message	server version 17. 13\n This is the emule server! \n

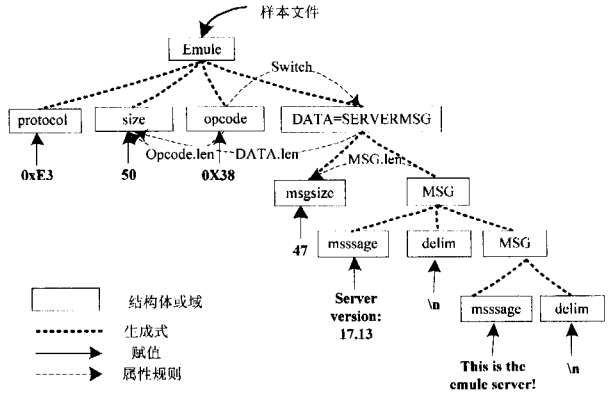

图 4 Emule 协议服务器消息报文的文法分析树

3.2 测试脚本的自动生成

文法分析树是样本数据的抽象图形表示，要实现测试脚本的自动化生成，只需遍历文法分析树，按照测试工具的测试逻辑，采用相应的脚本语言描述文法元素即可。以 SPIKE 为目标测试平台，算法 2 给出了根据文法分析树生成其测试脚本的具体生成流程。

该算法为 SPIKE 测试脚本生成算法的递归表示，以构造的文法分析树 G 和需要 Fuzz 测试的字段集合为输入，对文法分析树进行先序遍历：首先访问树的根节点 root(第 1～8 行)；再初始化变量 G_i 为树的最左子树(第 6 行)；然后从左至右依次访问 G 的子树(第 7～9 行)。在访问节点时，根据该节点类型建立不同的脚本描述：

1)若为叶子节点(第 2 行)，需要判断该节点是否为被测字段。若是被测字段(第 3，4 行)，则依据该节点的类型属性建立字段变量标签(如 s_string_variable 等)，并将其值属性作为字段初始值；若不是，则建立字段常量标签(如 s_binary 等)，并将其值属性作为字段默认值。

2)若为中间节点即结构体，则根据 MakeBlock 函数构造块标签，即访问起始建立 s_block_start("*")标识，当该中间节点的所有子节点访问完全，则采用 s_ block_end("*")闭合块标签。若有长度约束关系，则用 s_block_size_binary_bigendian_word( )描述长度字段。

算法 2 测试脚本生成算法

输入：文法生成树 G，测试字段集合 test；
输出：SPIKE 测试脚本

```
ScriptGen (Tree G) {
1. v←G. root
```

```
2. if(v is Leaf node)
3.   if(v∈test)
4.     Script←VarableElement(v);
5.   else
6.     Script←ConstantElement(v);
7. else
8.   Script←MakeBlock(v);
9.   Gi=leftsome_child(v);
10.  while Gi!=NULL do
11.    ScriptGen(Gi);
12.    Gi=right_sibling(Gi);
13. return Script;
14. }
```

4 测试方法与结果分析

为了验证上述方法的有效性，本文实现了自动脚本生成工具 FSG(Fuzzing Script Generator)，采用 XML 语言描述文法模型，可自动生成 SPIKE 测试脚本。以 Emule 协议作为实验对象，通过网络抓包工具截获发往客户端的 20 个报文作为样本，共生成 20 个测试脚本。以表 1 中的服务器消息报文为例，FSG 自动生成的 SPIKE 测试脚本如下：

```
s_block_start("emule");
s_binary("e3");
s_binary_block_size_word_bigendian_variable("data");
s_block_start("data");
s_binary("38");
s_block_start("servermsg");
s_binary_block_size_halfword_bigendian_variable("message");
s_block_start("message0");
s_string_variable("server version:17.13");
s_string_variable("\n");
s_block_end("message0");
s_block_start("message1");
s_string_variable("this is the emule server!");
s_string_variable("\n");
s_block_end(message1);
s_block_end(message);
s_block_start("servermsg");
s_block_start("data");
s_block_start("emule");
```

将自动生成的 20 个测试脚本作为 SPIKE 的输入，以 Emule 0.42 的客户端程序为测试目标，模拟服务器程序对其进行测试。表 2 给出了每个测试脚本的测试结果。

在测试过程中，结合双向切片技术[11]对触发异常的关键代码进行定位，并将关键代码相同的异常进行合并。对得到的 4 类异常进行分析，验证了已公布的 3 个已知漏洞并发现了程序中存在的 1 个未知漏洞。其中 2 个已知漏洞(BAGTRAQ-8445，BAGTRAQ-8440)是由于客户端对服务器端返回的消息中服务器名处理不当造成的字符串溢出漏洞，另 1 个已知漏洞(BAGTRAQ-8443)是客户端处理来自服务器的带有格式字符串的服务器消息报文时触发的格式化字符串处理漏洞。未知漏洞是搜索结果报文(0x16)中存在的堆溢出漏洞。

表 2 实验测试结果

脚本编号	报文类型	测试用例数	异常数
1	服务器消息(0x38)	55390	30
2	ID 改变(0x40)	15639	0
3	服务器状态(0x34)	6324	0
4	服务器列表(0x32)	23663	0
5	服务器消息(0x38)	34235	15
6	服务器信息(0x41)	45692	23
7	源列表(0x42)	14547	0
8	源列表(0x42)	24533	2
9	状态回应(0x97)	16489	20
10	消息拒绝(0x05)	8981	2
11	搜索结果(0x16)	23731	20
12	搜索结果(0x16)	9023	17
13	ID 改变(0x42)	17639	0
14	服务器列表(0x32)	11542	0
15	服务器状态(0x34)	7003	0
16	服务器描述(0x99)	13820	33
17	状态回应(0x97)	15338	16
18	搜索结果(0x16)	30127	26
19	源列表(0x42)	24533	0
20	服务器列表(0x32)	15639	0

由实验结果可以看出，FSG 能够针对不同的报文格式生成有效的测试脚本，实现对目标程序的自动化 Fuzzing 测试。值得说明的是，本文方法实现了格式描述与测试逻辑的分离，因此只需设计对应的脚本生成算法便能重定向到其他 Fuzzing测试平台，具有较好的通用性。

结束语 本文提出了一种基于模型的模糊测试脚本自动生成方法，即采用高阶属性文法构建抽象的数据格式模型，并将样本输入转化为带格式知识的文法分析树，最终实现测试脚本的自动化生成。下一步工作将考虑利用逆向分析技术[12]自动化生成文法模型，以进一步降低人工参与程度。

参 考 文 献

[1] 陈韬，孙乐昌，潘祖烈，等. 基于文件格式的漏洞挖掘技术研究[J]. 计算机科学，2011，38(10)：78-82

[2] Zhu Xue-yong, Wu Zhi-yong. A New Fuzzing Method Using Multi Data Samples Combination[J]. Journal of Computers, 2011,6(5):881-888

[3] The Advantages of Block-Based Protocol Analysis for security testing[R]. Aitel, D. Immunity Inc., 2002

[4] SPIKE[CP/OL]. http://www.immuitysec.com, 2012-04-23

[5] Peach[CP/OL]. http://www.peachFuzzer.com, 2012-04-23

[6] Sulley[CP/OL]. www.fuzzing.org/sulleymanual, 2012-04-23

[7] 沈亚楠，赵荣彩，王小芹，等. 基于文件规范描述的文件模糊测试[J]. 计算机工程，2010，36(16)：52-54

[8] Knuth E D. Semantics of context-free languages[J]. Mathematical System Theory, 1968, 2(2): 127-145

[9] Vogt H H, Swierstra S D, Kuiper M F, et al. Higher-order Attribute Grammars[J]. ACM SIGPLAN Notices, 1989, 24(7): 131-145

[10] Emule[CP/OL]. http://www.emule-project.net, 2012-04-27

[11] 于璐，吴礼发，庄洪林，等. 基于双向切片的软件漏洞分析技术研究[J]. 微电子学与计算机，2011，28(8)：173-175

[12] 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，2011，34(2)：242-255