**无线物联网漏洞检测**

**——模糊测试分析文档**

2020年11月25日

### 目 录

[目 录 2](#_Toc57212390)

[1 模糊测试概要原理 3](#_Toc57212391)

[1.1 模糊测试简介 3](#_Toc57212392)

[1.2 网络协议模糊测试 3](#_Toc57212393)

[1.3 模糊测试基本过程 4](#_Toc57212394)

[1.4 测试方法分类 5](#_Toc57212395)

[2 Spike模糊测试框架分析 6](#_Toc57212396)

[2.1 Spike介绍 6](#_Toc57212397)

[2.2 Spike用法 6](#_Toc57212398)

[2.3 Spike测试分析 8](#_Toc57212399)

[3 Fuzzowski模糊测试框架分析 11](#_Toc57212400)

[3.1 Fuzzowski简介 11](#_Toc57212401)

[3.2 Fuzzowski功能介绍 11](#_Toc57212402)

[3.3 Fuzzowski协议实现 12](#_Toc57212403)

[3.4 Fuzzowski用法 12](#_Toc57212404)

[3.5 Fuzzowski测试分析 15](#_Toc57212405)

1. 模糊测试概要原理
   1. 模糊测试简介

模糊测试（Fuzzing），是一种通过向目标系统提供非预期的输入并监视异常结果来发现软件漏洞的方法。

其核心思想是自动或半自动的生成随机数据输入到一个程序中，并监控目标程序异常，如崩溃，断言(assertion)失败，以发现可能的程序错误，比如内存泄漏等。

即用随机坏数据（也称做 fuzz）攻击一个程序，然后等着观察哪里遭到了破坏。但这一过程却能揭示出程序中的重要 bug。

它是一种介于完全的手工渗透测试与完全的自动化测试之间的安全性测试类型。它充分利用了机器能够随机生成和发送数据的能力。

* 1. 网络协议模糊测试

对网络协议进行模糊测试也需要识别出可被攻击的接口，通过变异或生成方式得到能够触发错误的模糊测试值，然后将这些模糊测试值发送给目标应用，监视目标应用的错误。

目前 ,最常见的网络协议模糊测试实施方案有两种：

客户端和服务端测试模式，即模糊器和被测对象分别为测试过程的两个端点。此时, 模糊器可充当客户端的角色，用来测试服务端程序的安全性, 例如 Web 服务程序。同时 ,模糊器也可以充当服务端的角色 ,用来测试客户端程序的安全性。模糊器中的监控模块用来对被测对象的行为进行收集、分析以判断是否存在异常情况。

网络协议模糊测试的实施方案是为了测试防火墙、路由器、安全网关等等部署在网络中间的设备。模糊器构造的数据被发送至协议服务器的过程中, 位于模糊器和协议服务器之间的被测对象对其起到了重组和解析的作用 ,一旦重组和解析过程中出错,可能造成被测对象出现异常状态。模糊器中的监控模块用来对被测对象的异常状态进行收集、分析,最终定位漏洞所在。通过此方法可发现被测对象在网络协议处理过程中的安全漏洞。

* 1. 模糊测试基本过程

1）确定测试的目标

2）确定输入的向量

3）生成模糊测试数据，可由测试工具通过随机或是半随机的方式生成

4）执行模糊数据测试，将生成的数据发送给被测试的系统（输入）

5）检测被测系统的状态（如是否能够响应，响应是否正确等）

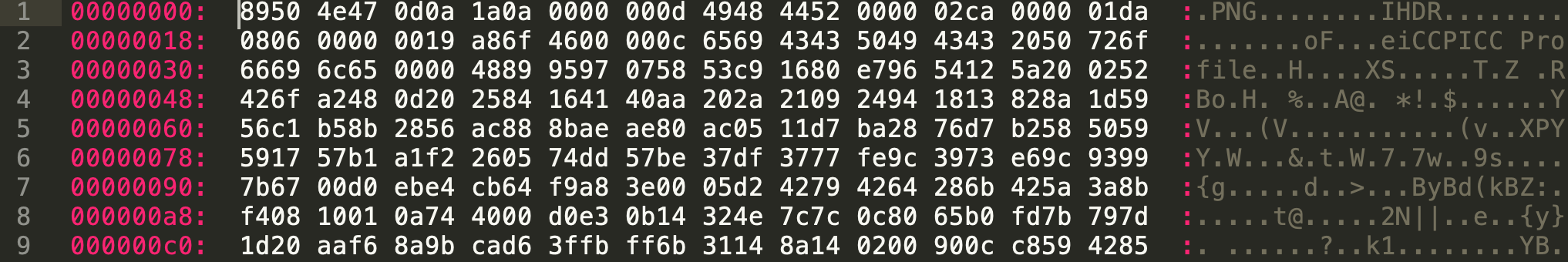
6）判定发现的漏洞是否可能被利用

* 1. 测试方法分类

基于变异的模糊测试——简而言之就是正常调用协议，抓包，然后混淆数据包达到生成异常数据包的结果，从而进行测试。

这种方法对已有的正常数据集依赖较高。需要有足够丰富的合法输入从而产生足够丰富的测试类型。

例如，png图片除了文件头后面数据内容进行置换混淆得到异常测试数据。

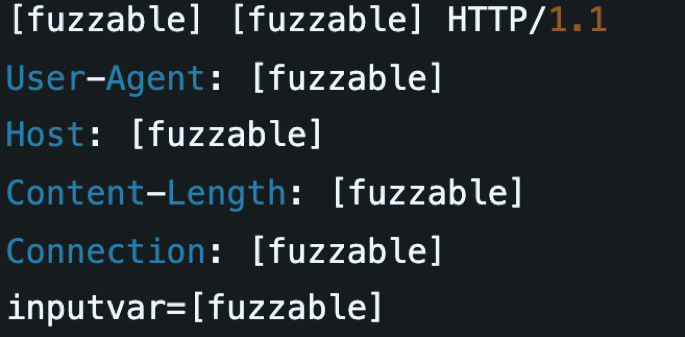


基于生成的模糊测试——简而言之就是理解协议规约定义，创建文法自动生成动态模糊的测试用例。

这种方法对协议的理解掌握程度需求更高。难度更大。

http的post请求如图，其中fuzzable的点可用来生成测试例子。





1. Spike模糊测试框架分析
   1. Spike介绍

从技术层面上讲，SPIKE是一个模糊测试创建工具集，它允许用户用C语言基于网络协议生成他们自己的测试数据。SPIKE定义了一些原始函数，这允许C程序员可以构建fuzzing数据向目标服务器发送，以期引起产生错误。   
 SPIKE有块（blocks）的概念，这可以在SPIKE内部推测出指定部分的大小。产生的数据就可以被SPIKE以不同的格式嵌入到自身测试数据中去。当需要在特定位置嵌入精确数据的时候，块大大节省了我们自行计算的时间。   
 SPIKE用模糊字符串库中的内容迭代模糊变量，达成模糊测试。模糊字符串可以是任何数据类型，甚至是XDR编码的二进制数据数组。SPIKE是一个GPL的API和一套工具，它使你可以快速创建任何网络协议压力测试的测试器。大多数协议都是围绕着非常类似的数据格式化建立的。这些协议中的许多都已经在SPIKE中得到支持。

* 1. Spike用法

1）SPIKE API的工作方式：

独特的SPIKE数据结构支持长度和分块s\_block\_start()，s\_block\_end()，s\_blocksize\_halfword\_bigendian()。SPIKE实用例程使得处理二进制数据，网络代码和常用的编组例程变得简单S\_xdr\_string()。SPIKE模糊框架自动重复所有可能有潜在问题的地方s\_string(“Host: “);s\_tring\_vayiable(“localhost”)。

2）SPIKE的数据结构：

SPIKE是一种先进先出队列，或称为“Buffer Class”，SPIKE能够自动填充长度域：S\_size\_string(“post”,5)；S\_block\_start(“Post”)；S\_string\_variable(“user=bob”)；S\_block\_end(“post”)。

3）一些基本调用：

s\_binary("00 00"); 插⼊⼆进制数据

s\_string("Referer\r\n"); 插⼊⼀个字符串常量

s\_int\_variable(0x00,3); 插⼊⼀个int变量

s\_string\_variable("test\r\n"); 插⼊⼀个字符串变量

s\_string\_variables('&',"username=bob&password=feet");插⼊多个字符串变量

s\_block\_start("block1");

s\_block\_end("block1"); 表示⼀个数据块的开始和结束

* 1. Spike测试分析

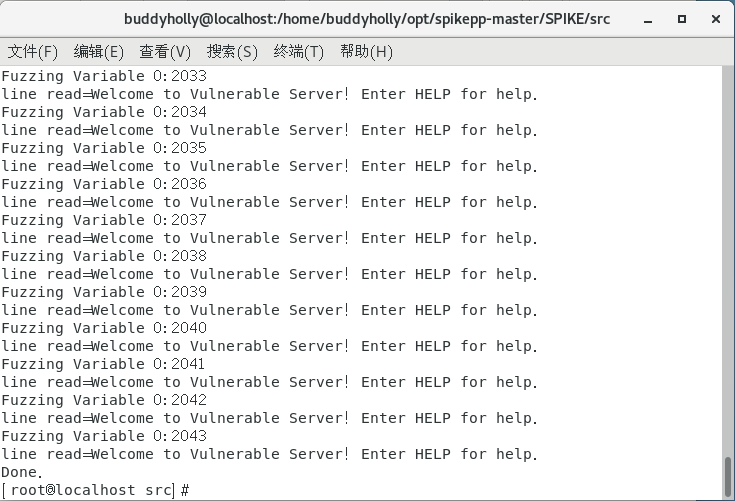
1. Vulnserver程序测试

通过使⽤vulnserver，测试spike能否正确使⽤。 脚本如下：

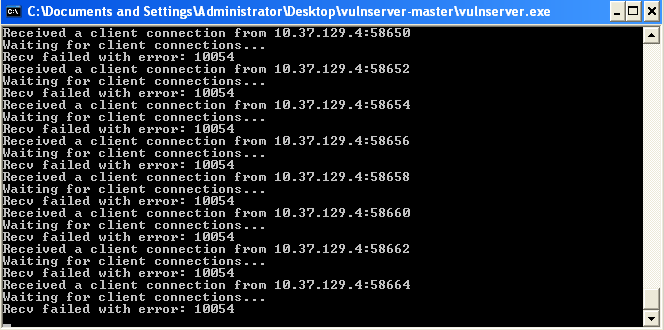
1. s\_readline(); s\_string("TRUN");
2. s\_string\_variable("COMMAND");

攻击端（CentOS）：

1. ./generic\_send\_tcp 10.xx.xx.xx 9999 vul\_test.spk 0 0



服务器端（WindowsXP）程序崩溃：



1. Web测试

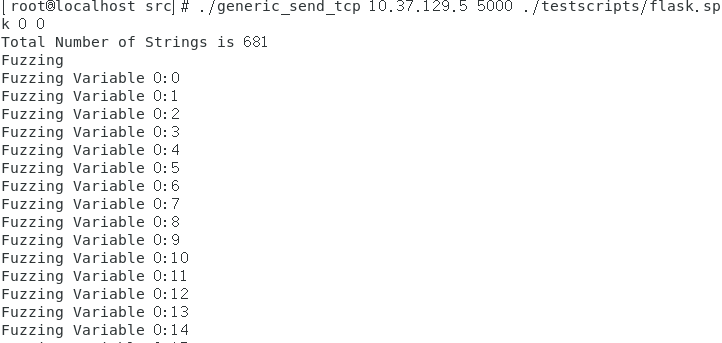
搭建一个WEB服务器测试spike使用。

攻击脚本如下：

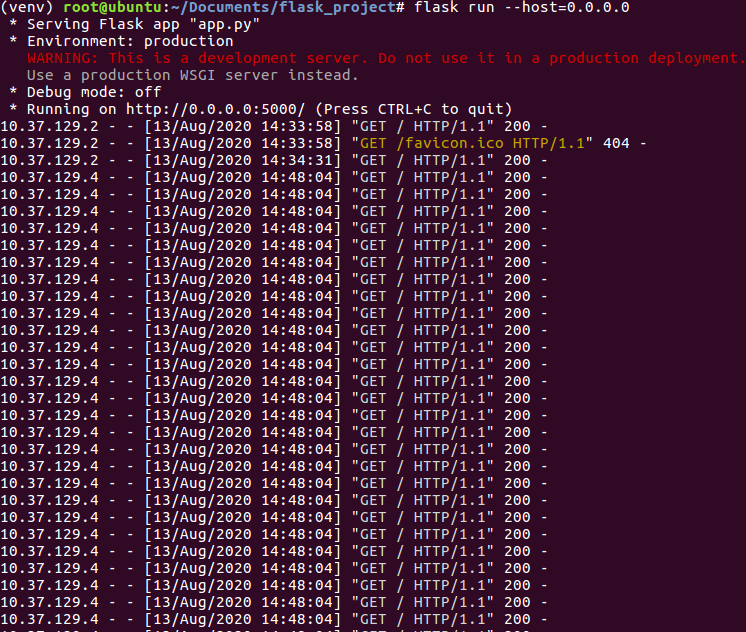
1. s\_string("GET / HTTP/1.1\r\n");
2. s\_string("Host: 10.37.129.5:5000\r\n");
3. s\_string("User-Agent: ");
4. s\_string("Content-Length: ");
5. s\_blocksize\_string("block1", 5);
6. s\_string("\r\nConnection: close\r\n\r\n");
7. s\_block\_start("block1");
8. s\_string("inputvar=");
9. s\_block\_end("block1");

攻击端（CentOS）：

1. ./generic\_send\_tcp 10.37.129.5 5000 flask.spk 0 0



服务器端（Ubuntu）：



1. Fuzzowski模糊测试框架分析
   1. Fuzzowski简介

Fuzzowski的设计核心理念，就是想让任何一个网络安全从业人员都会第一选择去使用它，该工具可以帮助研究人员对网络协议进行模糊测试，并且能够在整个测试过程中给我们提供帮助。除此之外，该工具还允许研究人员定义链接，并帮助识别服务的崩溃。

* 1. Fuzzowski功能介绍

1）基于Sulley Fuzzer实现数据收集功能【GitHub传送门】

2）基于BooFuzz部分功能【GitHub传送门】

3）Python3

4）非随机性

5）需要指定创建数据包的类型（SPIKE fuzzer风格）

6）允许使用参数创建元数据包，可指定注入点

7）提供功能强大的命令行终端

8）允许跳过引起错误的参数

9）自动化

10）提供完整可视化的可疑数据包内容

11）可将PoC存储为Python脚本

12）提供监控模块帮助实现数据收集

* 1. Fuzzowski协议实现

1）LPD(Line Printing Daemon)：完整实现

2）IPP (Internet Printing Protocol)：部分实现

3）BACnet(Building Automation&Control networks Protocol)：部分实现

4）Modbus(ICS communication protocol)：部分实现

* 1. Fuzzowski用法

1) 进入python虚拟环境：source venv/bin/activate

2) 进入fuzzowski的shell界面：

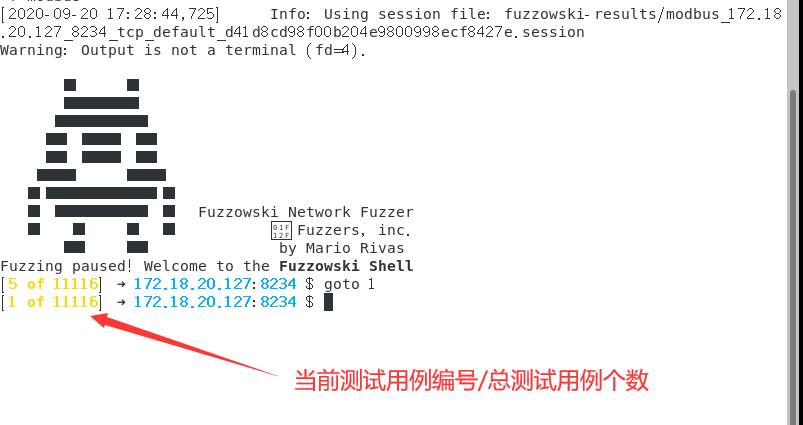
python3 -m fuzzowski host port [-p [{tcp,udp}] -f {modbus,...}

例如对网关进行测试：

python3 -m fuzzowski 172.18.20.127 8234 -p tcp -f modbus

（参数的详细含义可通过python3 -m fuzzowski -h 命令查阅 ）

进入了shell之后的界面：



连按两次tab键可显示所有可用命令：



3) 进行模糊测试

常用命令：

test命令：使用当前测试用例进行一次测试，不进行fuzzing

fuzz命令：使用当前测试用例进行一次测试，进行fuzzing

continue命令：从当前测试用例开始依次进行测试，每个用例都进行fuzzing

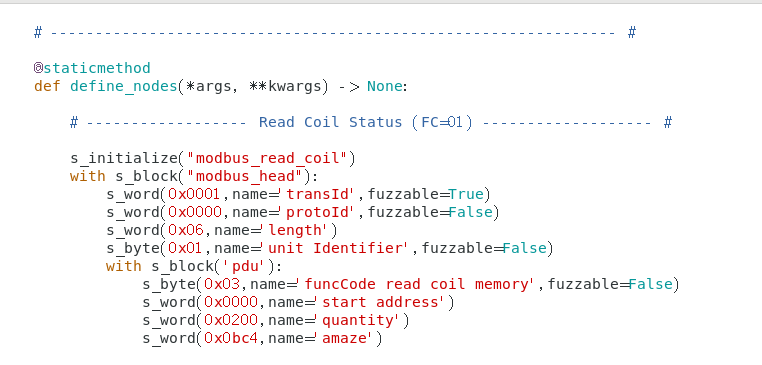
print命令：显示当前测试用例内容

poc命令：显示发送当前测试用例的程序

4) 修改测试用例内容

如要对测试用例的内容进行修改，例如修改某一字段的值，或这设置是否允许某一字段的值被fuzzing，可修改。

fuzzowski-master/fuzzowski/fuzzer/modbus/modbus.py文件



其中s\_word和s\_byte函数可设置modbus命令的字段，设置fuzzable的值即可将其设定为是否允许被fuzzing。

5) 查看测试日志

测试的记录存放在fuzzowski-master/fuzzowski-results目录下

6）工具使用样例

使用默认参数，对get\_printer\_attribs IPP操作进行模糊测试：

python -m fuzzowski printer1 631 -f ipp -r get\_printer\_attribs --restart smartplug

使用IPP的元功能来对指纹协议（Finger Protocol）进行模糊测试：

python -m fuzzowski printer 79 -f raw -r '{{root}}\n'

使用IPP的元功能来对指纹协议（Finger Protocol）进行模糊测试，但使用的是一个文件：

python -m fuzzowski printer 79 -f raw -r '{{root}}\n' --file 'path/to/my/fuzzlist'

* 1. Fuzzowski测试分析

1）搭建socket环境

1. **import** socket
3. HOST = '0.0.0.0'
4. PORT = 6678
5. count = 1
7. s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)
8. s.bind((HOST, PORT))
9. s.listen(10)
11. **print**('Server start at: %s:%s' %(HOST, PORT))
12. **print**('wait for connection...')
14. **while** True:
15. conn, addr = s.accept()
16. data = conn.recv(1024)
18. f = open("test1.txt",'a')
19. f.write("{:<5d}:".format(count))
21. **for** c **in** data:
22. f.write('%02X' % ord(c))
23. f.write(' ')
25. f.write('\n')
26. f.close()
28. **print**("{:<5d}:Connected by{}".format(count, addr))
30. count+=1
31. conn.send(data)
32. conn.close()

2）Fuzzowski设置

使用python3 -m fuzzowski 127.0.0.1 6789 -f modbus

经过输出测试，发现变动一个word型会产生140种十六进制数据，变动一个byte型会产生116种十六进制数据，为固定产生非随机生成。

通过测试modbus的read\_coil模块，主要为/fuzzowski-master/fuzzowski/fuzzers/modbus/modbus.py中如下代码：

1. s\_initialize("modbus\_read\_coil")
2. with s\_block("modbus\_head"):
3. s\_word(0x0001,name='transId',fuzzable=True)
4. s\_word(0x0000,name='protoId',fuzzable=False)
5. s\_word(0x06,name='length')
6. s\_byte(0xff,name='unit Identifier',fuzzable=False)
7. with s\_block('pdu'):
8. s\_byte(0x01,name='funcCode read coil memory',fuzzable=False)
9. s\_word(0x0000,name='start address')
10. s\_word(0x0000,name='quantity')
12. s\_initialize('read\_holding\_registers')
13. with s\_block("modbus\_head"):
14. s\_word(0x0001,name='transId',fuzzable=False)
15. s\_word(0x0002,name='protoId',fuzzable=False)
16. s\_word(0x06,name='length')
17. s\_byte(0xff,name='unit Identifier',fuzzable=False)
18. with s\_block('read\_holding\_registers\_block'):
19. s\_byte(0x01,name='read\_holding\_registers')
20. s\_word(0x0000,name='start address')
21. s\_word(0x0000,name='quantity')

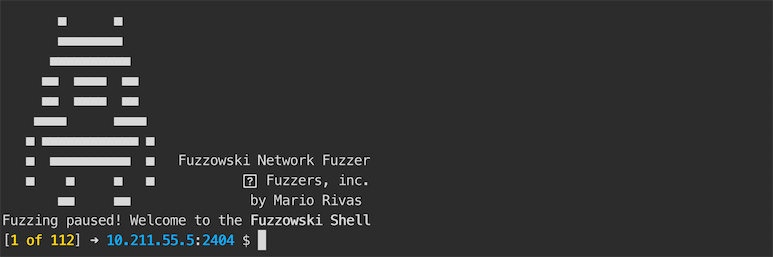
其中，fuzzable默认为True。设定为True的字段会依次产生变动生成数据，而非交叉产生，即一个数据包仅会有一个字段被fuzz。

举例修改上述模块为：

1. s\_initialize("modbus\_read\_coil")
2. with s\_block("modbus\_head"):
3. s\_byte(0x68,name='funcCode read coil memory1',fuzzable=False)
4. s\_byte(0x04,name='funcCode read coil memory2',fuzzable=False)
5. s\_byte(0x07,name='funcCode read coil memory3',fuzzable=True)
6. s\_byte(0x00,name='funcCode read coil memory4',fuzzable=False)
7. s\_byte(0x00,name='funcCode read coil memory5',fuzzable=False)
8. s\_byte(0x00,name='funcCode read coil memory6',fuzzable=False)
10. s\_initialize('read\_holding\_registers')
11. with s\_block("modbus\_head"):
12. s\_word(0x0001,name='transId',fuzzable=False)
13. # s\_word(0x0002,name='protoId',fuzzable=False)
14. # s\_word(0x06,name='length')
15. # s\_byte(0xff,name='unit Identifier',fuzzable=False)
16. # with s\_block('read\_holding\_registers\_block'):
17. #     s\_byte(0x01,name='read\_holding\_registers')
18. #     s\_word(0x0000,name='start address')
19. #     s\_word(0x0000,name='quantity')

即只会生成68 04 07 [00] 00 00其中会对第4字节的fuzz数据，共112条。

在示例图片中可以看到，经过修改了自动生成的测试数据变为112条，经过输出对比发现按照内置变异规则生成了需要的测试数据。



通过python3 -m fuzzowski 127.0.0.1 6789 -f modbus -tn -rt 1 -r read\_coil调用modbus的read\_coil模块（实际上只是生成测试数据发送tcp包，设置为不处理响应）。

通过Fuzzowski的单双字节模糊测试变异，通过分析进而辅助生成101、104协议所相关的测试数据用以后续测试。