

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
1 模糊测试概要原理	3
1.1 模糊测试简介	3
1.2 网络协议模糊测试	3
1.3 模糊测试基本过程	4
1.4 测试方法分类	5
2 Spike 模糊测试框架分析	6
2.1 Spike 介绍	6
2.2 Spike 用法	6
2.3 Spike 测试分析	8
3 Fuzzowski 模糊测试框架分析	11
3.1 Fuzzowski 简介	11
3.2 Fuzzowski 功能介绍	11
3.3 Fuzzowski 协议实现	12
3.4 Fuzzowski 用法	12
3.5 Fuzzowski 测试分析	15

1 模糊测试概要原理

1.1 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。

1.2 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。

1.3 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

1.4 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进行置换混淆得到异常测试数据。

```
1 00000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452 0000 02ca 0000 01da :.PNG.....IHDR.....
2 00000018: 0806 0000 0019 a86f 4600 000c 6569 4343 5049 4343 2050 726f :.....oF...eICCPICC Pro
3 00000030: 6669 6c65 0000 4889 9597 0758 53c9 1680 e796 5412 5a20 0252 :file..H....XS....T.Z .R
4 00000048: 426f a248 0d20 2584 1641 40aa 202a 2109 2494 1813 828a 1d59 :Bo.H. %..A@. *!.$.....Y
5 00000060: 56c1 b58b 2856 ac88 8bae ae80 ac05 11d7 ba28 76d7 b258 5059 :V...(V.....(v..XPY
6 00000078: 5917 57b1 a1f2 2605 74dd 57be 37df 3777 fe9c 3973 e69c 9399 :Y.W...&.t.W.7.7w..9s...
7 00000090: 7b67 00d0 ebe4 cb64 f9a8 3e00 05d2 4279 4264 286b 425a 3a8b :{g....d..>...ByBd(kBZ:..
8 000000a8: f408 1001 0a74 4000 d0e3 0b14 324e 7c7c 0c80 65b0 fd7b 797d :.....t@.....2N| |..e..{y}
9 000000c0: 1d20 aaf6 8a9b cad6 3ffb ff6b 3114 8a14 0200 900c c859 4285 :. ....?.k1.....YB.
```

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。

```
POST /testme.php HTTP/1.1
User-Agent: Mozilla/4.0
Host: testserver.example.com
Content-Length: 256
Connection: close
inputvar=admin
```

```
[fuzzable] [fuzzable] HTTP/1.1
User-Agent: [fuzzable]
Host: [fuzzable]
Content-Length: [fuzzable]
Connection: [fuzzable]
inputvar=[fuzzable]
```

2 Spike 模糊测试框架分析

2.1 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。

2.2 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_string_variable("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

2.3 Spike 测试分析

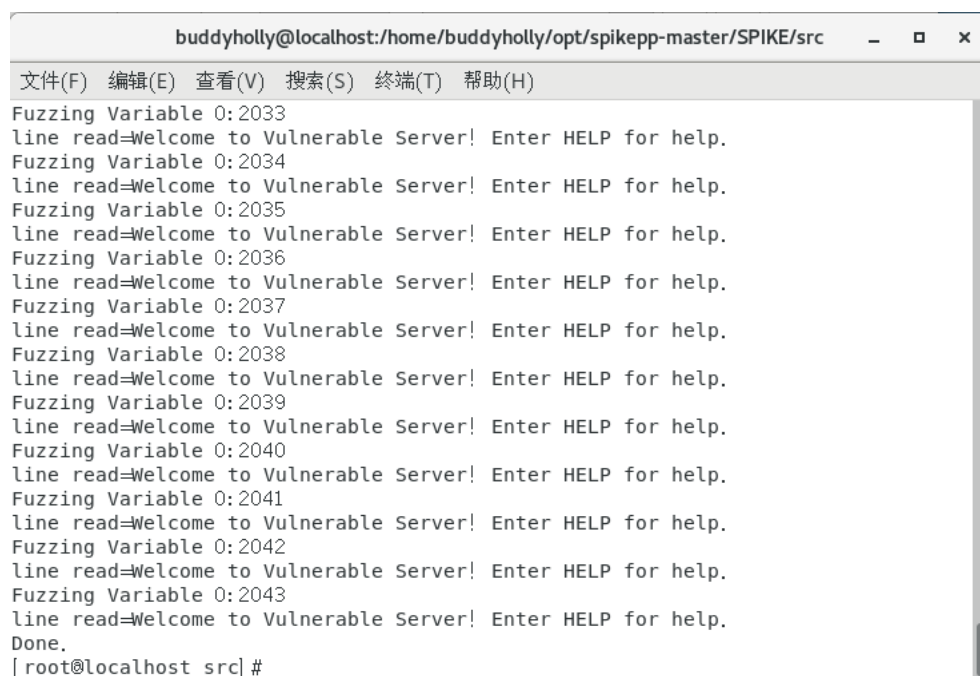
1) Vulnserver 程序测试

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");
```

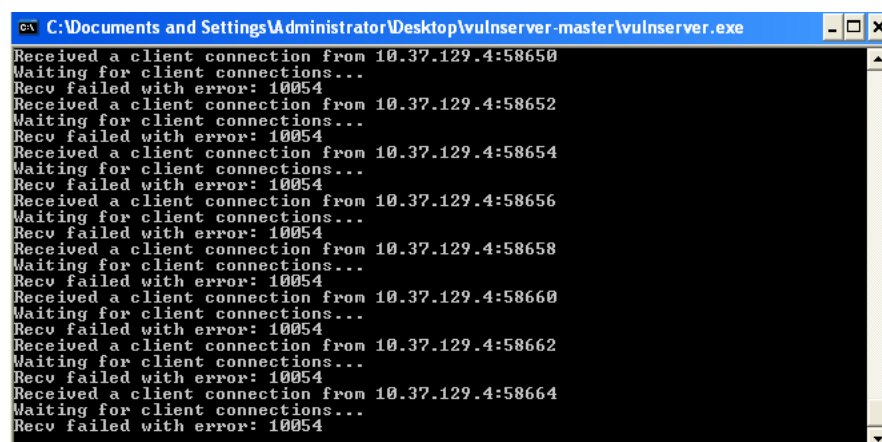
攻击端（CentOS）：

```
1. ./generic_send_tcp 10.xx.xx.xx 9999 vul_test.spk 0 0
```



```
buddyholly@localhost:/home/buddyholly/opt/spikepp-master/SPIKE/src  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
Fuzzing Variable 0:2033  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:2034  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:2035  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:2036  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:2037  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:2038  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:2039  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:2040  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:2041  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:2042  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:2043  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Done.  
[root@localhost src] #
```

服务器端（WindowsXP）程序崩溃：



```
C:\Documents and Settings\Administrator\Desktop\vulnserver-master\vulnserver.exe  
Received a client connection from 10.37.129.4:58650  
Waiting for client connections...  
Recv failed with error: 10054  
Received a client connection from 10.37.129.4:58652  
Waiting for client connections...  
Recv failed with error: 10054  
Received a client connection from 10.37.129.4:58654  
Waiting for client connections...  
Recv failed with error: 10054  
Received a client connection from 10.37.129.4:58656  
Waiting for client connections...  
Recv failed with error: 10054  
Received a client connection from 10.37.129.4:58658  
Waiting for client connections...  
Recv failed with error: 10054  
Received a client connection from 10.37.129.4:58660  
Waiting for client connections...  
Recv failed with error: 10054  
Received a client connection from 10.37.129.4:58662  
Waiting for client connections...  
Recv failed with error: 10054  
Received a client connection from 10.37.129.4:58664  
Waiting for client connections...  
Recv failed with error: 10054
```


2) 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
```

```
|root@localhost src| # ./generic_send_tcp 10.37.129.5 5000 ./testscripts/flask.spk 0 0
Total Number of Strings is 681
Fuzzing
Fuzzing Variable 0:0
Fuzzing Variable 0:1
Fuzzing Variable 0:2
Fuzzing Variable 0:3
Fuzzing Variable 0:4
Fuzzing Variable 0:5
Fuzzing Variable 0:6
Fuzzing Variable 0:7
Fuzzing Variable 0:8
Fuzzing Variable 0:9
Fuzzing Variable 0:10
Fuzzing Variable 0:11
Fuzzing Variable 0:12
Fuzzing Variable 0:13
Fuzzing Variable 0:14
```

服务器端（Ubuntu）：

[illegible]

3 Fuzzowski 模糊测试框架分析

3.1 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。

3.2 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

3.3 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

3.4 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:

```
[2020-09-20 17:28:44,725] Info: Using session file: fuzzowski-results/modbus_172.18.20.127_8234_tcp_default_d41d8cd98f00b204e9800998ecf8427e.session
Warning: Output is not a terminal (fd=4).
```

```
Fuzzowski Network Fuzzer  
by Mario Rivas  
Fuzzing paused! Welcome to the Fuzzowski Shell  
[5 of 11116] → 172.18.20.127:8234 $ goto 1  
[1 of 11116] → 172.18.20.127:8234 $
```

当前测试用例编号/总测试用例个数

连接两次 **tab** 键可显示所有可用命令:

```

# ----- #

@staticmethod
def define_nodes(*args, **kwargs) -> None:

    # ----- Read Coil Status (FC=01) ----- #

    s_initialize("modbus_read_coil")
    with s_block("modbus_head"):
        s_word(0x0001, name='transId', fuzzable=True)
        s_word(0x0000, name='protoId', fuzzable=False)
        s_word(0x06, name='length')
        s_byte(0x01, name='unit Identifier', fuzzable=False)
    with s_block('pdu'):
        s_byte(0x03, name='funcCode read coil memory', fuzzable=False)
        s_word(0x0000, name='start address')
        s_word(0x0200, name='quantity')
        s_word(0x0bc4, name='amaze')

```

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_attris` IPP 操作进行模糊测试：

```
python -m fuzzowski printer1 631 -f ipp -r get_printer_attris --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'
```

3.5 Fuzzowski 测试分析

1) 搭建 socket 环境

```
1. import socket
2.
3. HOST = '0.0.0.0'
4. PORT = 6678
5. count = 1
6.
7. s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8. s.bind((HOST, PORT))
9. s.listen(10)
10.
11. print('Server start at: %s:%s' %(HOST, PORT))
12. print('wait for connection...')
13.
14. while True:
15.     conn, addr = s.accept()
16.     data = conn.recv(1024)
17.
18.     f = open("test1.txt", 'a')
19.     f.write("{:<5d}:".format(count))
20.
21.     for c in data:
22.         f.write('%02X' % ord(c))
23.         f.write(' ')
24.
25.     f.write('\n')
26.     f.close()
27.
28.     print("{:<5d}:Connected by{}".format(count, addr))
29.
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')
11.
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)
```



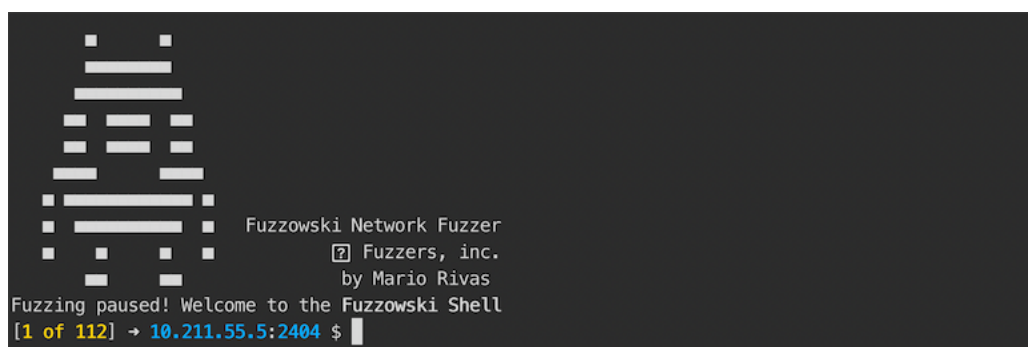
```

8.     s_byte(0x00,name='funcCode read coil memory6',fuzzable=False)
9.
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。