Sulley fuzzer learning---1

行之 / 2018-08-13 20:50:45 /

前言

这一系类的文章是对Sulley fuzzer的学习，文章的开始部分是对Sulley使用手册的翻译，后续的还有Sulley的实际应用。
手册链接：http://www.fuzzing.org/wp-content/SulleyManual.pdf

Sulley: Fuzzing Framework

Sulley是一个可开发的fuzzer和多个可扩展组件组成的fuzz框架。恕我直言，Sulley无论从商业还是公共领域所表现的能力来看都超出了以前发布的大多数fuzz技术。该框架Inc.(怪兽电力公司这部电影)的主角亲切地命名，well，因为它有绒毛（fuzzy）

在大多数情况下，现代的fuzzers仅专注于数据生成。Sulley不仅数据生成的能力让人印象深刻，而且更进一步地包括了一个现代fuzzer应该提供的很多其他重要方面的信息。

- Sulley可以条不紊地监视网络并保留记录。
- Sulley可以检测和监控目标的健康状况，能够使用多种方法恢复到已知的良好状态。
- Sulley可以检测，跟踪和分类检测到的故障。
- Sulley可以并行地fuzz，显著地提高测试速度。
- Sulley可以自动确定测试用例触发的特殊错误。
- Sulley自动完成了这一切，而且不需要再有人动手。
  Sulley的总体用法分解如下：
- Data Representation：使用任何fuzzer的第一步。 在抓住数据包的同时运行目标并勾选一些接口。
  将协议分解为单独的请求，并将其表示为Sulley中的block。
- Session：将您的请求链接在一起形成一个session，附加各种可用的Sulley监控代理（网络，调试器等）并开始fuzzing。
- Post Mortem：查看生成的数据和监控结果。 重现个别测试用例。

Authors

- Pedram Amini, pamini@tippingpoint.com
- Aaron Portnoy, aportnoy@tippingpoint.com

Documentation

- SulleyDirectoryStructure
- SulleyInstallation
- SulleyDataRepresentation
- SulleySessions
- SulleyPostMortem
- SulleyWalkthroughTrend
- SulleyDevNotes

Sulley Directory Structure

Sulley目录结构有一些排版理由。维护目录结构将确保使用Legos,requests,utilities扩展fuzzer时一切都能够保持得井井有条。以下概述了您需要了解的目录结构（目录以粗
archived_fuzzies: 这是一个由fuzz目标名称组成的目录(Free)，用来存储归档的fuzzers以及从fuzz session生成的数据。

- trend_server_protect_5168:这种已停用的fuzz会在本文档后面的逐步介绍中被引用。
- trillian_jabber: :文档中引用的另一个已停用的fuzz。

audits:应将已记录的PCAP，崩溃数据，代码覆盖范围和active fuzz seesion的分析情况保存到此目录中。
一旦被停用，记录的数据将会被移至'archived_fuzzies'。
docs:本文档和生成的Epydoc API参考。
requests:Sulley的需求库。 每个目标都应该有自己的文件，可以用来存储多个请求。

- REQUESTS.html：此文件包含存储的请求类别的说明，并列出了各种类型。 以字母顺序。
- http.py：各种Web服务器fuzzing测试请求。
- trend.py：包含与本文档后面讨论的完整的fuzz演练相关的请求。

sulley：fuzzer框架。 除非您想扩展框架，否则您不需要触接触这些文件。

- legos：用户定义的复杂基元:
  ber.py:ASN.1 / BER原语
  cerpc.py:Microsoft RPC NDR原语
  misc.py: 各种未分类的复杂原语，如电子邮件地址和主机名
  xdr.py:XDR类型
- pgraph：Python图形抽象库。 在建立session中使用。
- utils：各种帮助程序。
  dcerpc.py：Microsoft RPC帮助程序例程，例如绑定到接口和生成请求。
  misc.py：各种未分类的例程，例如CRC-16和UUID操作例程。
  scada.py：SCADA特定的辅助程序，包括DNP3block编码器。
- init.py：此处定义了用于创建请求的各种's_'别名。
- blocks.py：Blocks和Blocks helps在这里定义。
- pedrpc.py：此文件定义了Sulley用于各种代理和主要的fuzzer之间通信的客户端和服务器类。
- primitives.py：这里定义了各种fuzzer原语，包括静态，随机，字符串和整数。
- sessions.py：用于构建和执行会话的功能

  sex.py:Sulley的自定义异常处理类

  unit_tests: Sulley's unit testing harness.

- utils：各种独立的实用程序。
  ida_fuzz_library_extend.py：一个IDApython脚本，它静态搜索二进制文件以获取扩展fuzz库的值。
  crashbin_explorer.py：用于浏览存储在序列化崩溃bin文件中的结果的命令行实用程序

WikiStart
pcap_cleaner.py：用于清除与故障无关的所有条目的PCAP目录的命令行实用程序。
network_monitor.py:PedRPC网络监控代理驱动。
process_monitor.py:PedRPC基于调试器的目标监视代理程序的驱动。
unit_test.py:Sulley的单元测试线束。
vmcontrol.py:PedRPCVMWare控制代理驱动。

Data Representation

Aitel had it right with
SPIKE，我们已经仔细研究了每一个我可以得到的fuzzer，并且Sulley基于协议的block表示方法比其他的方法更简单，同时也代表了大多数协议的灵活性。Sulley利用基于b

```
s_initialize("new request")
```

现在，您开始向request添加基元，block和嵌套block。每个基元都可以单独渲染和变异。渲染原语以原始数据格式返回其内容。变异原语会转换其内部内容。渲染和变异的
developers中抽象出来的，所以不要担心。但是要知道当fuzzable values耗尽时，每个可变基元都会接受一个默认值.

Static and Random Primitives

让我们从最简单的原语s_static（）开始，它为请求添加任意长度的静态不可变异值。Sulley中有各种别名，s_dunno（），s_raw（）和s_unknown（）都是s_static（）的

```
# these are all equivalent:■■■■■■■■■■■■
s_static("pedram\x00was\x01here\x02")
s_raw("pedram\x00was\x01here\x02")
s_dunno("pedram\x00was\x01here\x02")
s_unknown("pedram\x00was\x01here\x02")
```

基元，block等都采用可选的关键字名称作为参数。指定名称允许您通过request.namesname直接从请求访问命名项，而不必遍历block结构以到达所需元素。

与上述相关但不相同的是s_binary（）原语，它接受以多种格式表示的二进制数据。 SPIKE用户将识别此API，其功能（或者应该）与您已熟悉的相同:

```
# yeah, it can handle all these formats.■■■■■■■■■■■■■■■■■■■■
s_binary("0xde 0xad be ef \xca fe 00 01 02 0xba0xdd f0 0d", name="complex")
```

Sulley的大多数原语都是由"fuzz heuristics"驱动的，因此具有有限数量的变异。 一个例外是s_random（）原语，它可用于生成不同长度的随机数据。
该原语采用两个必需参数'min_length'和'max_length'，分别指定在每次迭代时生成的随机数据的最小和最大长度。 该原语还接受以下可选关键字参数：

- num_mutations: (integer, default=25) 恢复为默认值之前要进行的变异次数。
- fuzzable: (boolean, default=True) 启用或禁用此原语的fuzzing测试。
- name: (string, default=None) 与指定名称的所有Sulley对象一样，您可以在整个请求中直接访问此原语。

通常使用分隔符将字符串解析为子字段。 例如，空格字符用作HTTP请求"GET /index.html HTTP / 1.0"中的分隔符。
同一请求中的前斜杠（/）和点（。）字符也是分隔符。 在Sulley中定义协议时，请务必使用s_delim（）原语表示分隔符。
与其他原语一样，第一个参数是必需的，用于指定默认值。 与其他原语一样，s_delim（）接受选项'fuzzable'和'name'关键字参数。
分隔符突变包括重复，替换和排除。 作为一个完整的示例，请考虑以下基元序列来模糊HTMLbody标签。

```
# fuzzes the string: <BODY bgcolor="black">
s_delim("<")
s_string("BODY")
s_delim(" ")
s_string("bgcolor")
s_delim("=")
s_delim("\"")
s_string("black")
s_delim("\"")
s_delim(">")
```

Fuzz Library Extensions

Sulley的原语包含一个内部的"fuzz library"，一个潜在的有趣的循环值列。 如果您不想破解源文件来扩展字符串和/或整数的fuzz library，那么您可以轻松地进行外部操作,您只需在要启动模糊驱动程序的目录中创建.fuzz_strings或.fuzz_ints文件即可。 将每个模糊值放在它自己的行上。 Sulley将使用这些文件中的值在运行时更新原始库。

Blocks

掌握了原语之后，让我们接下来看看它们如何组织和嵌套在block中。 使用closeslock_start（）定义和打开新block，并使用closeslock_end（）关闭。 必须为每个block指定一个名称，并将其指定为closeslock_start（）的第一个参数。 此例程还接受以下可选关键字参数：

- group: (string, default=None) 与此block关联的组的名称，稍后将对此进行更多说明。
- encoder: (function pointer, default=None)指向函数的指针，用于将呈现的数据传递到返回之前。
- dep: (string, default=None) 可选原语，其特定值取决于此block。
- dep_value: (mixed, default=None) 对于要呈现的block，字段"dep"必须包含的值。
- dep_values: (list of mixed types, default=[]) 字段"dep"的值可以包含要呈现的block。
- dep_compare (string, default="==")应用于依赖的比较方法。 有效选项包括："==", "!=", ">", ">=", "<" and "<=".

分组，编码和dependencies是大多数其他框架中没有的强大功能，值得进一步剖析。关于block的重要而简单的注释。 block关闭后无法更新例如：

```
block-a:
integer
string
size(block-b)
block-b:
integer
Integer
```

适用于block-b的block-a中的sizer将永远不会在上述场景中呈现，这是因为当block关闭时，所有基元的渲染被bubbled up到block。
所以当block-b关闭并更新sizer时，它实际上并没有bubbled up。
这种限制只是暂时的，在将来可以得到解决。在这种情况下，简单地将sizer放在block-a之外即可。

Groups

Groupping允许您将block绑定到group原语，以指定block应循环遍历group中每个值的所有可能变异。
例如，group原语用于表示具有类似参数结构的有效操作码或动词的列表。 原语s_group（）定义一个group并接受两个必需参数。
第一个指定group的名称，第二个指定要迭代的可能原始值的列表。 举一个简单的例子，考虑以下完整的Sulley请求，用于fuzz Web服务器：

```
# import all of Sulley's functionality.
from sulley import *
# this request is for fuzzing: {GET,HEAD,POST,TRACE} /index.html HTTP/1.1
# define a new block named "HTTP BASIC".
s_initialize("HTTP BASIC")
# define a group primitive listing the various HTTP verbs we wish to fuzz.
s_group("verbs", values=["GET", "HEAD", "POST", "TRACE"])
# define a new block named "body" and associate with the above group.
if s_block_start("body", group="verbs"):
# break the remainder of the HTTP request into individual primitives.
s_delim(" ")
s_delim("/")
s_string("index.html")
s_delim(" ")
s_string("HTTP")
s_delim("/")
s_string("1")
s_delim(".")
s_string("1")
# end the request with the mandatory static sequence.
```

```
s_static("\r\n\r\n")
# close the open block, the name argument is optional here.
s_block_end("body")
```

该脚本首先导入Sulley的所有组件。接下来，初始化新请求并命名为“HTTP
BASIC”。稍后可以引用此名称以直接访问此请求。接下来，使用名称“verbs”和可能的字符串值“GET”，“HEAD”，“POST”和“TRACE”定义group。使用名称“body”启动一个session中时，模糊器将为block “body”生成并传输所有可能的值，对于组中定义的每个动词一次。请注意，group可以独立使用，不必绑定到特定block。

Encoders

Encoders是一个简单但功能强大的block修饰符。可以指定函数并将其附加到block，以便在返回和通过线路传输之前修改该block的变异内容。
这里有一个很好的真实例子来解释。 Trend Micro ConTrol
Manager的DcsProcessor.exe守护程序侦听TCP端口20901，并期望接收使用专有XOR编码例程格式化的数据。
对decoder进行逆向工程，开发了以下XOR编码程序：

```
def trend_xor_encode (str):
key = 0xA8534344
ret = ""
# pad to 4 byte boundary.
pad = 4 - (len(str) % 4)
if pad == 4:
pad = 0
str += "\x00" * pad
while str:
dword = struct.unpack("<L", str[:4])[0]
str = str[4:]
dword ^= key
ret += struct.pack("<L", dword)
key = dword
return ret
```

Sulley Encoder采用一个单一的参数，要编码的数据并返回编码后数据。这个定义的Encoder现在可以连接到一个包含fuzzable基元的block，允许fuzzer developer继续，好像这个小障碍永远不存在(This defined encoder can
now be attached to a block containing fuzzable primitives allowing the fuzzer developer to continue as if this little hurdle never existed.)。

Dependencies

Dependencies允许您将条件应用于整个block的渲染。 这是通过首先将block链接到原语来实现的，它将依赖于使用可选的'dep'关键字参数。
当Sulley渲染dependant block时，它将检查链接原语的值并相应地表现。 可以使用'dep_value'关键字参数指定相关值。
或者，可以使用'dep_values'关键字参数指定相关值列表。 最后，可以通过'dep_compare'关键字参数修改实际的条件比较。
例如，考虑一种情况，根据整数的值，预期会有不同的数据：

```
s_short("opcode", full_range=True)
# opcode 10 expects an authentication sequence.
if s_block_start("auth", dep="opcode", dep_value=10):
s_string("USER")
s_delim(" ")
s_string("pedram")
s_static("\r\n")
s_string("PASS")
s_delim(" ")
s_delim("fuzzywuzzy")
s_block_end()
# opcodes 15 and 16 expect a single string hostname.
if s_block_start("hostname", dep="opcode", dep_values=[15, 16]):
s_string("pedram.openrce.org")
s_block_end()
# the rest of the opcodes take a string prefixed with two underscores.
if s_block_start("something", dep="opcode", dep_values=[10, 15, 16], dep_compare="!="):
s_static("__")
s_string("some string")
s_block_end()
```

Block dependencies可以以任何数量的方式链接在一起，从而允许强大的（但却很复杂）组合。

Block Helpers

在数据生成方面，要有效利用Sulley必须熟悉的是block helpers。 这包括sizer，校验和(checksums)和中继器(repeaters)。

Sizers

SPIKE用户将熟悉s_sizer（）（或s_size（））block helper。 block helper测量块名称来测量的大小作为第一个参数，并接受以下其他关键字参数：

- length :(整数，默认= 4）size字段长度。
- endian :(character，default ='<'）位字段的Endianess。 为小端指定'<'，为大端指定'>'。
  格式:(字符串，默认=“二进制”）输出格式，“二进制”或“ascii”，控制整数的格式
  原语呈现。
- inclusive :(boolean，default = False）sizer应该计算自己的长度吗？
- signed :(boolean，default = False）使sizers有符号或无符号，仅在format =“ascii”时适用。
- math：（functon，default = None）将此函数中定义的数学运算应用于size。
- fuzzable :(boolean，default = False）启用或禁用此基元的fuzz。
- name :(string，default = None）与所有Sulley对象一样，指定名称使您可以在整个请求中直接访问此原语。

Sizer是数据生成中的关键组件，允许表示复杂协议，如XDR表示法，ASN.1等。Sulley将在渲染sizer时动态计算相关block的长度。 默认情况下，Sulley不会fuzz size字段。 在许多情况下，这是期望的行为，但是，如果不是，则启用fuzzable。

Checksums

与sizer类似，s_checksum（）帮助程序使用块名称来计算作为第一个参数的校验和。还可以指定以下可选关键字参数：

algorithm :(字符串或函数指针，默认=“crc32”）。 校验和算法应用于目标block。（crc32，adler32，md5，sha1）

endian :(character，default ='<'）位字段的编码模式。 为小端指定'<'，为大端指定'>'。

length :(整数，默认= 0）校验和的长度，保留为0表示自动计算。

name :(string，default = None）与所有指定名称的Sulley对象一样，您可以在整个请求中直接访问此原语。
'algorithm'参数可以是“crc32”，“adler32”，“md5”或“sha1”之一。 或者，您可以为此参数指定函数指针以应用自定义校验和算法。

Repeaters

s_repeat（）（或s_repeater（））helper用于复制块可变的次数。这对于再解析具有多个元素的表时进行溢出测试很有帮助。这个helper需要三个参数，即要重复的块的名
此外，还提供以下可选关键字参数：

- step :(integer ，default = 1）最小和最大代表之间的步数。
- fuzzable :(boolean，default = False）启用或禁用此基元的fuzz。
- name :(string，default = None）与指定名称的所有Sulley对象一样，您可以在整个请求中直接访问此原语。

请思考以下将三个helper绑定在一起的例子。 我们正在fuzz包含一个字符串表的协议的一部分。
表中的每个entry包括一个2字节的字符串类型字段，一个2字节的长度字段，一个字符串字段，最后是一个在字符串字段上计算的CRC-32校验和字段。
我们不知道类型字段的有效值是什么，因此我们将使用随机数据进行fuzz。 以下是在Sulley中的协议的一部分：

```
# table entry: [type][len][string][checksum]
if s_block_start("table entry"):
# we don't know what the valid types are, so we'll fill this in with random data.
s_random("\x00\x00", 2, 2)
# next, we insert a sizer of length 2 for the string field to follow.
s_size("string field", length=2)
# block helpers only apply to blocks, so encapsulate the string primitive in one.
if s_block_start("string field"):
# the default string will simply be a short sequence of C's.
s_string("C" * 10)
s_block_end()
# append the CRC-32 checksum of the string to the table entry.
s_checksum("string field")
s_block_end()
# repeat the table entry from 100 to 1,000 reps stepping 50 elements on each iteration.
s_repeat("table entry", min_reps=100, max_reps=1000, step=50)
```

Sulle脚本不仅会fuzz表entry的输入解析，而且可能会发现在处理过长的表时出现的错误。

Legos

Sulley利用“Legos”来表示用户定义的组件，例如Microsoft RPC，XDR，ASN.1和电子邮件地址，主机名和协议原语等。在ASN.1 /
BER中，字符串表示为序列[0x04][0x84][dword length][string].。对基于ASN.1的协议进行fuzz时，每个字符串要包含前面的长度和类型前缀可能会变得很麻烦。
相反，我们可以定义Lego并引用它：

```
s_lego("ber_string", "anonymous")
```

每个Lego都遵循类似的格式，但可选的'options'关键字参数除外，该参数特定于各个legos。 举个简单的例子，考虑"标签"Lego的定义，在fuzz XML-ish协议时很有用：

```
class tag (blocks.block):
def __init__ (self, name, request, value, options={}):
blocks.block.__init__(self, name, request, None, None, None, None)
self.value = value
self.options = options
if not self.value:
raise sex.error("MISSING LEGO.tag DEFAULT VALUE")
#
# [delim][string][delim]
self.push(primitives.delim("<"))
self.push(primitives.string(self.value))
self.push(primitives.delim(">"))
```

这个Lego示例只是接受所需的tag作为字符串并将其封装在适当的分隔符中。它通过扩展块类并通过self.push（）手动将tag分隔符和用户提供的字符串添加到块堆栈来实现

这是另一个用于表示Sulley中的ASN.1 / BER整数的Lego的简单例子，选择"最小公分母"将所有整数表示为4字节整数，其格式如下：[0x02] [0x04] [dword]，其中0x02指定整数类型，0x04指定整数长度为4字节且'dword '代表我们传递的实际整数。以下是来自sulley \ legos \ ber.py的定义：

```
class integer (blocks.block):
def __init__ (self, name, request, value, options={}):
blocks.block.__init__(self, name, request, None, None, None, None)
self.value = value
self.options = options
if not self.value:
raise sex.error("MISSING LEGO.ber_integer DEFAULT VALUE")
self.push(primitives.dword(self.value, endian=">"))
def render (self):
# let the parent do the initial render.
blocks.block.render(self)
self.rendered = "\x02\x04" + self.rendered
return self.rendered
```

与前面的示例类似，使用self.push（）将提供的整数添加到block的堆栈中。 与前面的示例不同，render（）例程被重载以使呈现的内容前缀为静态序列"\ x02 \ x04"，以满足先前描述的整数表示要求。

Final Notes

Sulley随着每个新fuzzer的创造而成长。 开发的block/request扩展了request库，可以很容易地引用并用于构建将来的fuzzers。
有关更详细的API参考，请参阅Epydoc生成的Sulley API文档。