登录

Sulley fuzzer learning---4

行之 / 2018-08-20 11:10:57 / 浏览数 2331 技术文章 技术文章 顶(0) 踩(0)

前言

这里是Sulley使用手册的第三部分，从这里开始本系列文章将开始讲述Sulley fuzzer的实际应用。
手册链接：http://www.fuzzing.org/wp-content/SulleyManual.pdf

A Complete Walkthrough: Trend Micro Server Protect

为了更好地将这些东西联系在一起。我们将从头到尾完成一个实例的演示。这个例子将涉及许多中级到高级的Sulley概念，应该有助于巩固您对框架的理解。本节的主要目的
Micro Server Protect.具体来说是由SpntSvc.exe服务绑定的TCP端口5168上的Microsoft DCE / RPC端点。
RPC端点从TmRpcSrv.dll以下面的接口定义语言（IDL）stub信息公开：

```
// opcode: 0x00, address: 0x65741030
// uuid: 25288888-bd5b-11d1-9d53-0080c83a5c2c
// version: 1.0
error_status_t rpc_opnum_0 (
[in] handle_t arg_1, // not sent on wire
[in] long trend_req_num,
[in][size_is(arg_4)] byte some_string[],
[in] long arg_4,
[out][size_is(arg_6)] byte arg_5[], // not sent on wire
[in] long arg_6
);
```

参数'arg_1'和'arg_6'实际上都不是通过wire传输的。 当我们编写实际的fuzz
request时，这是一个需要考虑的重要情况。进一步检查发现参数'trend_req_num'具有特殊含义。此参数的上半部分和下半部分控制一对跳转表，可通过此单个RPC函数公
对跳转表进行逆向工程，出了下列组合：

```
When the value for the upper half is 0x0001, 1 through 21 are valid lower half values.
When the value for the upper half is 0x0002, 1 through 18 are valid lower half values.
When the value for the upper half is 0x0003, 1 through 84 are valid lower half values.
When the value for the upper half is 0x0005, 1 through 24 are valid lower half values.
When the value for the upper half is 0x000A, 1 through 48 are valid lower half values.
When the value for the upper half is 0x001F, 1 through 24 are valid lower half values.
```

接下来我们必须创建一个自定义编码器例程，它将负责将已定义的blok封装为有效的DCE / RPC请求。
我们定义了一个围绕utisl.dcerpc.request（）的基本包装器，它将操作码参数硬编码为零：

```
# dce rpc request encoder used for trend server protect 5168 RPC service.
# opnum is always zero.
def rpc_request_encoder (data):
return utils.dcerpc.request(0, data)
```

Building the Requests

有了这些信息和我们的编码器，我们就可以开始定义我们的Sulley requests了。 我们创建一个文件'requests \
trend.py'来包含我们所有与Trend相关的请求和helpers定义并开始编码.我们利用一些Python循环来自动为'trend_req_num'生成每个有效上限值的单独请求：

```
for op, submax in [(0x1, 22), (0x2, 19), (0x3, 85), (0x5, 25), (0xa, 49), (0x1f, 25)]:
s_initialize("5168: op-%x" % op)
if s_block_start("everything", encoder=rpc_request_encoder):
# [in] long trend_req_num,
s_group("subs", values=map(chr, range(1, submax)))
s_static("\x00") # subs is actually a little endian word
s_static(struct.pack("<H", op)) # opcode
# [in][size_is(arg_4)] byte some_string[],
s_size("some_string")
if s_block_start("some_string", group="subs"):
s_static("A" * 0x5000, name="arg3")
s_block_end()
# [in] long arg_4,
s_size("some_string")
# [in] long arg_6
```

```
s_static(struct.pack("<L", 0x5000)) # output buffer size
s_block_end()
```

在每个生成的请求中，初始化新block并传到我们先前定义的自定义编码器。接下来，s_group（）原语用于定义名为"subs"的序列，该序列表示我们之前看到的"trend_req
'trend_req_num'，因为我们已经逆向了它的有效值，如果我们没有，我们也可以fuzz这些字段。接下来，将'some_string'的NDR大小前缀添加到请求中。我们可以选择在
DCE / RPC NDR
Lego原语，但由于RPC请求非常简单，我们决定手动表示NDR格式。接下来，将'some_string'值添加到请求中。字符串值封装在一个块中，以便可以测量它的长度。在这个
我们的request现已准备就绪，我们可以继续创建session。

Creating the Session

我们在Sulley根文件夹中为我们的会话创建了一个名为"fuzz_trend_server_protect_5168.py"的新文件。当此文件已经完成其任务，会移至'archived_fuzzies'文件夹。首先
Sulley和创建的Trend request：

```
from sulley import *
from requests import trend
```

接下来，我们将定义一个预发送函数，该函数负责在传输任何测试用例之前建立DCE /
RPC连接。预发送例程接受单个参数，即传输数据的套接字。这是一个简单的编写例程，这要归功于utils.dcerpc.bind（）的可用性，这是一个Sulley实用程序例程：

```
def rpc_bind (sock):
   bind = utils.dcerpc.bind("25288888-bd5b-11d1-9d53-0080c83a5c2c", "1.0")
   sock.send(bind)
   utils.dcerpc.bind_ack(sock.recv(1000))
```

现在是时候启动session并定义target了。我们将fuzz一个目标，一个安装在VMWare虚拟机的Trend Server
Protect，地址为10.0.0.1。我们将遵循框架准则，将序列化的session信息保存到'audits'目录中。最后，我们用定义的target注册一个网络监视器、进程监视器和虚拟机控制

```
sess = sessions.session(session_filename="audits/trend_server_protect_5168.session")
target = sessions.target("10.0.0.1", 5168)
target.netmon = pedrpc.client("10.0.0.1", 26001)
target.procmon = pedrpc.client("10.0.0.1", 26002)
target.vmcontrol = pedrpc.client("127.0.0.1", 26003)
```

由于存在VMWare控制代理，因此无论何时检测到故障或无法到达target，Sulley都将默认恢复到正常的快照。如果VMWare控制代理程序不可用但进程监视器代理程可用

```
target.procmon_options = \
{
"proc_name" : "SpntSvc.exe",
"stop_commands" : ['net stop "trend serverprotect"'],
"start_commands" : ['net start "trend serverprotect"'],
}
```

每当您使用进程监视器代理时，'proc_name'参数都是必需的，它指定调试器应附加到哪个进程名称并查找其中的错误。如果VMWare控制代理和进程监视代理都不可用，那

接下来，我们通过调用VMWare控制代理程序restart_target（）例程来指示target启动。运行后，将target添加到session中，定义预发送例程，并将每个定义的request连

```
# start up the target.
target.vmcontrol.restart_target()
print "virtual machine up and running"
sess.add_target(target)
sess.pre_send = rpc_bind
sess.connect(s_get("5168: op-1"))
sess.connect(s_get("5168: op-2"))
sess.connect(s_get("5168: op-3"))
sess.connect(s_get("5168: op-5"))
sess.connect(s_get("5168: op-a"))
sess.connect(s_get("5168: op-1f"))
sess.fuzz()
```

Setting up the Environment

启动fuzz session之前的最后一步是设置环境。我们通过调出目标虚拟机映像并使用以下命令行参数直接在测试映像中启动网络和处理监视器代理来实现此目的：

```
network_monitor.py -d 1 \
-f "src or dst port 5168" \
-p audits\trend_server_protect_5168
process_monitor.py -c audits\trend_server_protect_5168.crashbin \
-p SpntSvc.exe
```

将BPF过滤器字符串传递给网络监视器，以确保仅记录我们感兴趣的数据包。还会选择audits文件夹中的目录，网络监视器将为每个测试用例创建PCAP。使用代理和目标进程
ready and waiting"。

接下来，我们关闭VMWare并在主机系统（fuzz测试系统）上启动VMWare控制代理。此代理程序需要vmrun.exe可执行文件的路径，要控制的实际映像的路径以及最终在

```
vmcontrol.py -r "c:\Progra~1\VMware\VMware~1\vmrun.exe" \
-x "v:\vmfarm\images\windows\2000\win_2000_pro-clones\TrendM~1\win_2000_pro.vmx" \
--snapshot "sulley ready and waiting"
```

Ready, Set ... Action! ... and post mortem.

最后，一切准备就绪。
只需启动'fuzz_trend_server_protect_5168.py'，将网络浏览器连接到http://127.0.0.1:26000即可监控fuzzer进度，然后坐下来观看并欣赏。

当fuzzer完成运行221个测试用例列表时，我们发现其中19个触发了故障。使用'crashbin_explorer.py'程序，我们可以查看由异常地址分类的错误：

```
$ ./utils/crashbin_explorer.py audits/trend_server_protect_5168.crashbin
[6] [INVALID]:41414141 Unable to disassemble at 41414141 from thread 568 caused access violation
42, 109, 156, 164, 170, 198,
[3] LogMaster.dll:63272106 push ebx from thread 568 caused access violation
53, 56, 151,
[1] ntdll.dll:77fbb267 push dword [ebp+0xc] from thread 568 caused access violation
195,
[1] Eng50.dll:6118954e rep movsd from thread 568 caused access violation
181,
[1] ntdll.dll:77facbbd push edi from thread 568 caused access violation
118,
[1] Eng50.dll:61187671 cmp word [eax],0x3b from thread 568 caused access violation
116,
[1] [INVALID]:0058002e Unable to disassemble at 0058002e from thread 568 caused access violation
70,
[2] Eng50.dll:611896d1 rep movsd from thread 568 caused access violation
152, 182,
[1] StRpcSrv.dll:6567603c push esi from thread 568 caused access violation
106,
[1] KERNEL32.dll:7c57993a cmp ax,[edi] from thread 568 caused access violation
165,
[1]Eng50.dll:61182415 mov edx,[edi+0x20c] from thread 568 caused access violation
```

其中一些显然是可利用的。例如，EIP为0x41414141的测试用例。测试用例70似乎偶然发现了一个可能的代码执行问题，一个UNICODE溢出（实际上在研究后这可能是一个
overflow）。Crash bin资源管理器实用程序还可以生成故障的图形视图，绘制堆栈回溯路径。这可以帮助确定某些问题的根本原因。
该实用程序接受以下命令行参数:

```
$ ./utils/crashbin_explorer.py
USAGE: crashbin_explorer.py <xxx.crashbin>
[-t|--test #] dump the crash synopsis for a specific test case number
[-g|--graph name] generate a graph of all crash paths, save to 'name'.udg
```

例如，我们可以在检测到故障时的CPU状态（测试用例＃70）：

```
$ ./utils/crashbin_explorer.py audits/trend_server_protect_5168.crashbin -t 70
[INVALID]:0058002e Unable to disassemble at 0058002e from thread 568 caused access violation
when attempting to read from 0x0058002e
CONTEXT DUMP
EIP: 0058002e Unable to disassemble at 0058002e
EAX: 00000001 ( 1) -> N/A
EBX: 0259e118 ( 39444760) -> A.....AAAAA (stack)
ECX: 00000000 ( 0) -> N/A
EDX: ffffffff (4294967295) -> N/A
EDI: 00000000 ( 0) -> N/A
ESI: 0259e33e ( 39445310) -> A.....AAAAA (stack)
EBP: 00000000 ( 0) -> N/A
ESP: 0259d594 ( 39441812) -> LA.XLT.......MPT.MSG.OFT.PPS.RT (stack)
+00: 0041004c ( 4259916) -> N/A
+04: 0058002e ( 5767214) -> N/A
+08: 0054004c ( 5505100) -> N/A
+0c: 0056002e ( 5636142) -> N/A
+10: 00530042 ( 5439554) -> N/A
+14: 004a002e ( 4849710) -> N/A
disasm around:
```

```
0x0058002e Unable to disassemble
SEH unwind:
0259fc58 -> StRpcSrv.dll:656784e3
0259fd70 -> TmRpcSrv.dll:65741820
0259fda8 -> TmRpcSrv.dll:65741820
0259ffdc -> RPCRT4.dll:77d87000
ffffffff -> KERNEL32.dll:7c5c216c
```

你可以在这里看到堆栈已经被看似是UNICODE文件扩展名的字符串所覆盖。 您也可以为给定的测试用例提取已存档的PCAP文件。

最后一步，我们希望删除所有不包含有关故障的信息的pCAP文件，'pcap_cleaner.py'实用程序是为完成此任务而编写的：

```
$ ./utils/pcap_cleaner.py
USAGE: pcap_cleaner.py <xxx.crashbin> <path to pcaps>
```

此实用程序将打开指定的crashbin文件，读入触发故障的测试用例编号列表并从指定目录中清除所有其他PCAP文件。
此fuzz中发现的代码执行漏洞均已报告给Trend，并给了以下建议：

- TSRT-07-01: Trend Micro ServerProtect StCommon.dll Stack Overflow Vulnerabilities (http://www.tippingpoint.com/security/advisories/TSRT-07-01.html)
- TSRT-07-02: Trend Micro ServerProtect eng50.dll Stack Overflow Vulnerabilities (https://www.trendmicro.com/en_us/business/products/network/intrusion-prevention.html)

这并不是说在这个接口中已经耗尽了所有可能的漏洞。 实际上，这是该接口最基本的fuzz测试。
实际上使用s_string（ ）原语而不是简单的长字符串的fuzz也可行。

点击收藏 | 0 关注 | 1
上一篇：沙箱逃逸 - Microsoft ... 下一篇：SpringMVC XStream...

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板