漏洞攻击可视化

自动化分析二进制漏洞攻击过程。

使用说明

适用目标

Linux下应用层的x86_64程序。不适用于与硬件设备高度相关的程序。

example

测试样例可见 `test/packed_*` ，其中 `run_visualization.py` 将直接运行一次完整的分析。
使用方法可参照 `run_visualization.py` 的内容。

记录功能

运行记录前，首先需要关闭系统的内存地址随机化。在项目根目录中运行该脚本即可。

```
./set-aslr.sh off
```

记录操作可以通过运行以下脚本完成：

```
./record.sh  target_binary
```

该脚本会应用装载器运行目标程序，以bash为例，可以通过以下方式记录一次运行：

```
./record.sh /bin/bash # 对/bin/bash进行记录
Writing logged output to bash_log
ls  #输入ls命令
bash_log  bin  doc  html  ls_log  maps.bash.98687  maps.ls.98675  maps.ls.98689
#ctrl^D 退出
```

在当前文件夹下会产生 `maps.target_name.pid` 与 `target_log` 文件，分别为记录的初始内存布局与外部输入。

重放功能

重放与分析部分建议在ipython的交互式环境下运行。

重放功能由 `source.replayer.Replayer` 实现。以下为以 `test/packed_heap_sample` 文件夹下的测试样例为例，导入包并根据记录的数据建立项目，并获取初始状态与模拟运行管理器。

```
# 首先导入文件
import sys
import os
sys.path.append("../../source/")
import parse_helpers
import replayer
import angr
import claripy
from imp import reload

# 根据记录的数据建立project
p = replayer.Replayer("easyheap", "./sample.txt", "maps.8998")

state = p.get_entry_state() # 获取初始状态
simgr = p.get_simgr() # 获取模拟运行管理器

simgr.run() # 进行一次完整的重放
```

重放时支持指定的攻击成功状态。以下为默认设置的execve钩子，目标程序调用execve系统调用，且第一个参数为 `/bin/sh` 时触发，成功触发设置的钩子时 `exploited_state` 会被设置，此状态表示攻击成功与结束，后续的所有分析均作用于程序启动到 `exploited_state` 之间。该钩子适用于大部分目标为getshell的攻击。

同样可以自定义一个钩子函数将任意的状态指定为攻击成功的标志，可钩取的目标有函数调用、系统调用、某指令地址或某部分内存被访问。可以根据 `state.regs` 与 `state.memory` 获取寄存器与内存信息。

```python
class exploited_execve(angr.SimProcedure):
    """
    Sample hook procedure to check if the programme is going to do `execve("/bi
    If the check pass, project.exploited_state will be set.
    """
    def run(self, filename, args, envp):
        print("run into execve!")

        # check if the first arg is like '/bin/sh'
        fname = self.state.memory.load(filename, 8)
        assert(fname.concrete)
        if b"sh" in hex2str(fname.args[0]):
            print("found exploited state")

            # set exploited_state, so we can get the final state from project
            assert(self.project and self.state)
            self.project.exploited_state = self.state
            # we don't need to continue
            self.exit(0)
        else:
            return claripy.BVV(0, 64)

    def __repr__(self):
        return '<exploited execve stub>'
```

分析功能

可供选择的分析子模块有：

- 函数调用分析：跟踪ROP攻击，以及捕获栈返回地址的溢出
- 堆内存分析：捕获堆中溢出以及堆上的异常操作
- got表分析：检查got表中函数指针是否存在异常（异常溯源将在之后的版本中实现），给出符号解析结果
- 信息泄漏分析：检查程序输出中是否存在内存地址信息，给出符号解析结果

分析子模块可根据目标程序的特点任意组合。可视化报告目前直接在终端中输出，堆变化图则生成在当前文件夹下。

```python
# 设置需要启动的分析模块
p.enable(["call_analysis", "heap_analysis", "leak_analysis", "got_analysis"])
# 开始分析
p.do_analysis()
```

开发接口

新增分析子模块只需要实现一个新类并为其实现 `do_analysis` 方法即可。

```
class X_analysis(analysis):
    def __init__(self, ):
        # do init
    def do_analysis(self):
        # do the work

register_analysis(X_analysis)
```