Silence

红日学院

分布式资产发现与漏洞扫描原理及实现

Silence

目录

个人介绍

- 姓名：Silence
- 职位：安全开发工程师
- 主要经历：
  - 18年6月，加入红日安全研发组
- 主要项目：
  - 《DVWA漏洞测试平台分析》
  - 《WebGoat漏洞测试平台分析》
  - 红日CTF比赛平台开发&维护
  - 启元学堂开发&维护
  - 红蓝训练营(vulnstack)开发

项目介绍

- GITHUB：https://github.com/imsilence/shadow-vulscan
- 资产发现：发现开放的服务信息，提供详细的资产指纹信息，如IP、OS、端口、服务等
  - 主动方式：IP扫描，端口扫描，指纹扫描，爬虫
  - 被动方式：流量，日志等
  - 工具：Nmap扫描，masscan等
- 漏洞扫描：监控&发现服务存在的安全风险，以便做到及时修复
  - 工具：burpsuite，awvs，zap，xssscan等
- 分布式：使用分布式方式解决单节点性能瓶颈及单节点故障等问题

项目介绍-组件

- 系统：CentOS 6.X
- 开发语言：Python3
- 第三方组件：Redis，PostgreSQL，nmap
- 开发框架：flask，SQLAlchemy

项目介绍-部署

- 安装源

  yum install epel-release

- 安装第三方组件

  yum install python3 redis postgresql-server nmap

- 准备postgresql数据库

  配置、initdb、用户/密码、库

- 准备py虚拟环境，安装第三方包：

  python3 -m venv venv && source venv/bin/activate && pip install -r requirements.all.txt

- 修改DB&redis配置

  shadow/config.py

项目介绍-部署

- 初始化数据库表

  bash run.sh shadow init-db

- 创建默认用户

  run.sh auth create-user --is_super

- 启动web服务

  nohup bash run.sh run --host 0.0.0.0 --port 8888 >/dev/null 2>&1 &

- 启动调度器

  nohup bash run.sh schedule start-schedule >/dev/null 2>&1 &

- 启动资产发现执行器

  nohup bash run.sh schedule start-execute --ident node01 --host 10.0.0.2 --port 8888 --type 2 --concurrent 5 >/dev/null 2>&1 &

项目介绍-部署

- 启动资产发现执行器

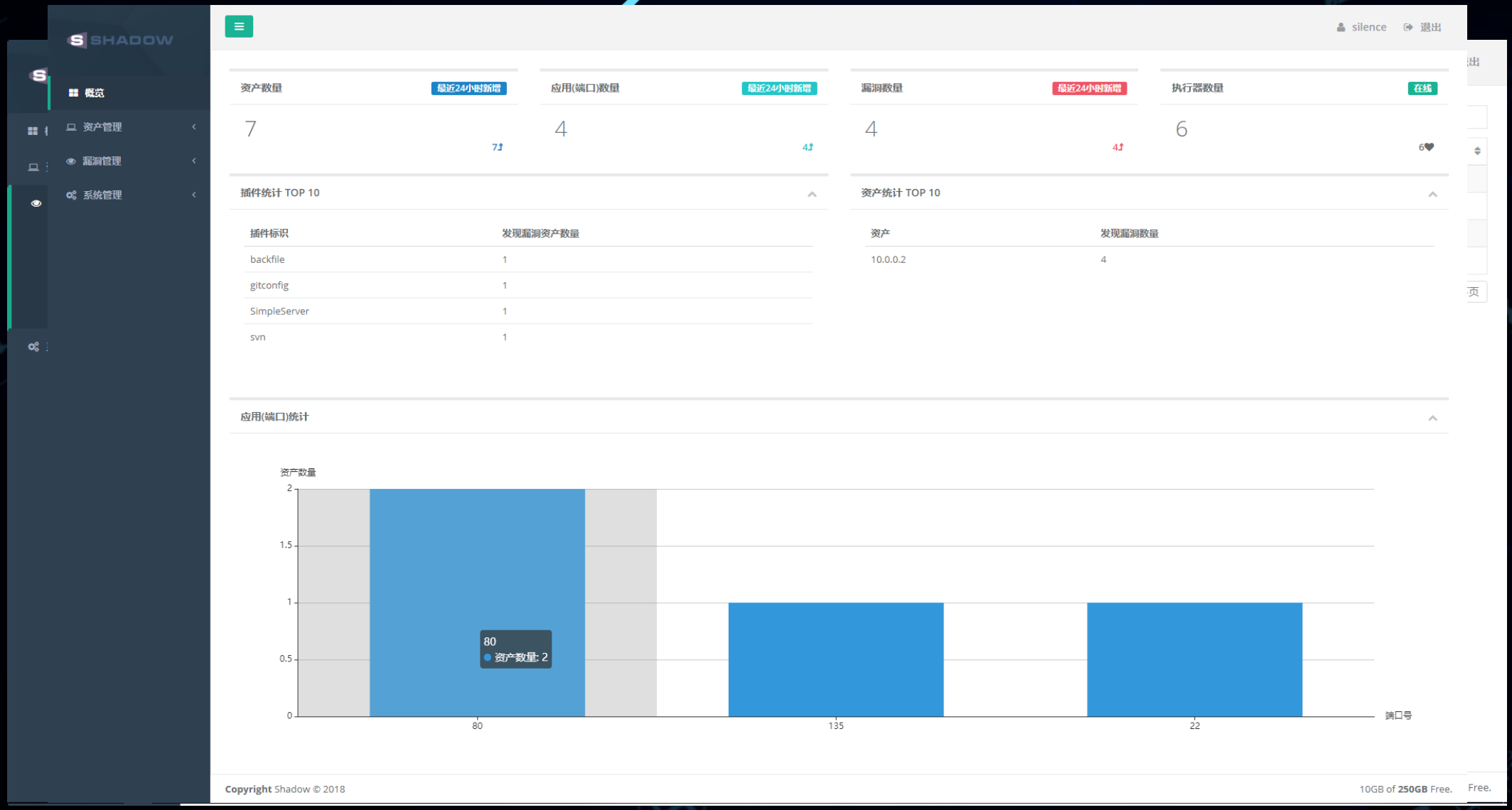
  nohup bash run.sh schedule start-execute --ident node01 --host 10.0.0.2 --port 8888 --type 2 --concurrent 5 >/dev/null 2>&1 &

- 启动漏洞扫描执行器

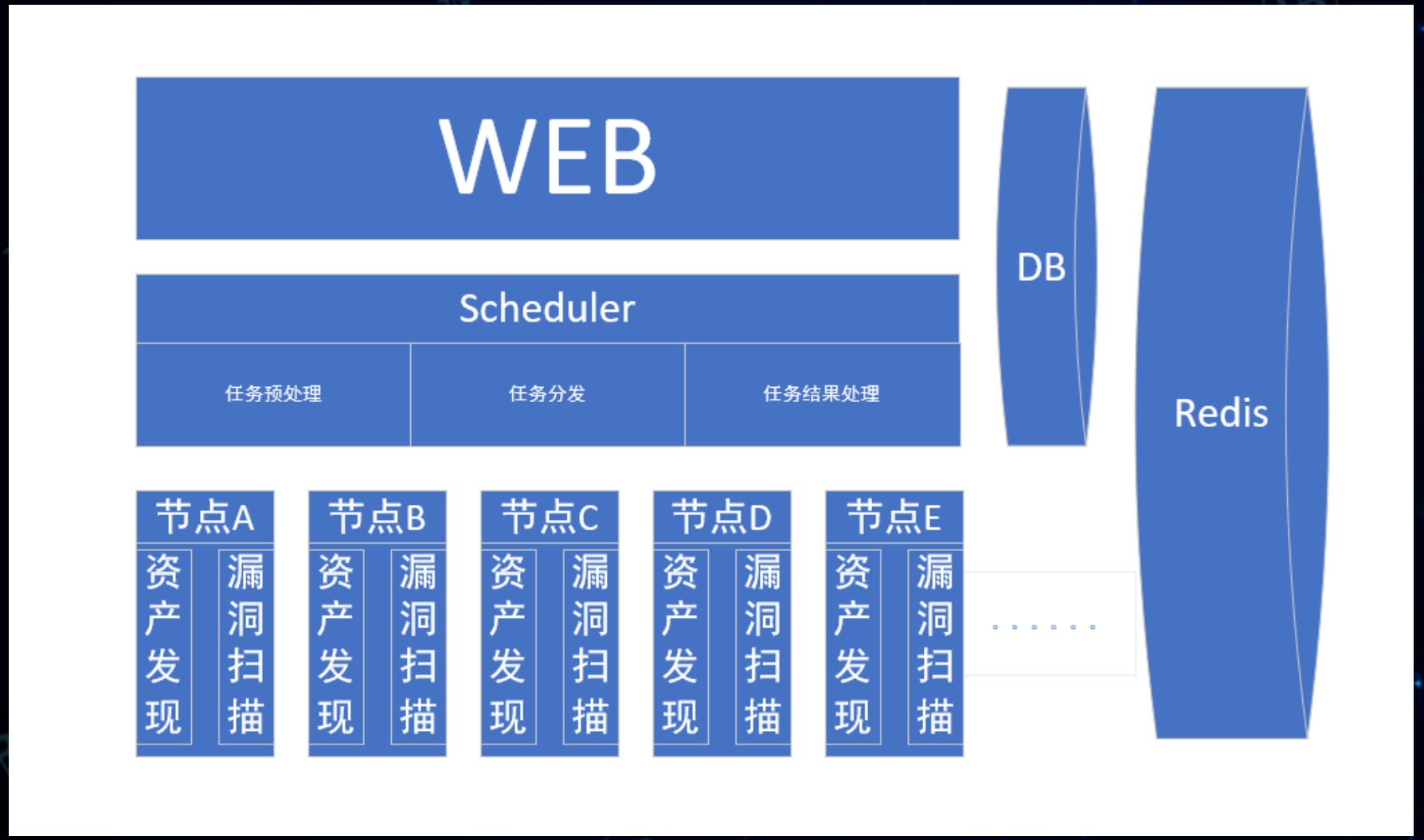
  nohup bash run.sh schedule start-execute --ident node01 --host 10.0.0.2 --port 8888 --type 3 --concurrent 5 >/dev/null 2>&1 &

- 执行器参数说明
  - ident: 进程标识(每台机器唯一)
  - host: web host
  - port: web port
  - type: 执行器类型，2:资产扫描进程，type: 3 漏洞扫描进程
  - concurrent: 执行线程数量

项目介绍-演示


SHADOW
概览
资产管理
漏洞管理
系统管理
silence
退出
资产数量
最近24小时新增
7
应用(端口)数量
最近24小时新增
4
漏洞数量
最近24小时新增
4
执行器数量
在线
6
插件统计 TOP 10
插件标识
发现漏洞资产数量
backfile 1
gitconfig 1
SimpleServer 1
svn 1
资产统计 TOP 10
资产
发现漏洞数量
10.0.0.2 4
应用(端口)统计
资产数量
2
1.5
1
0.5
0
80
资产数量: 2
135
22
端口号
Copyright Shadow © 2018
10GB of 250GB Free.

原理与架构

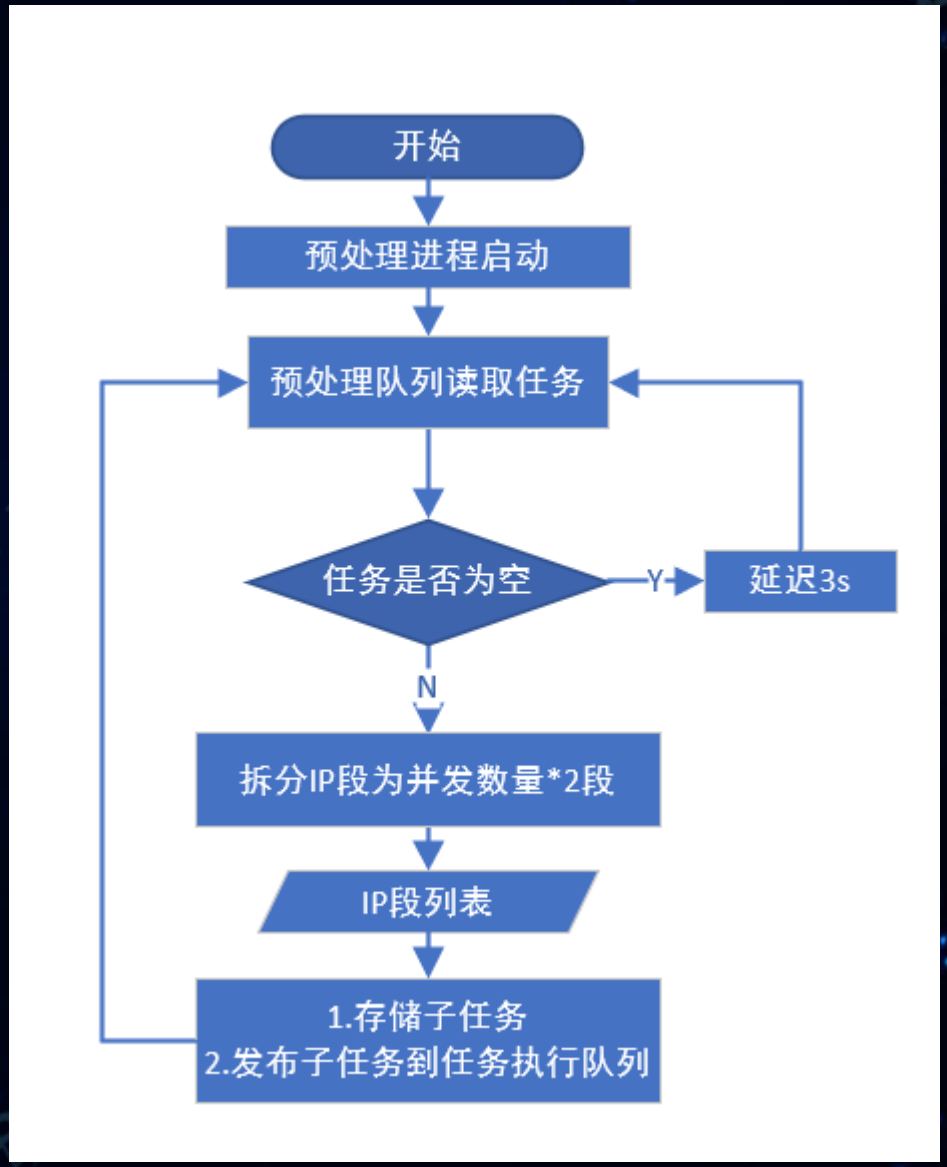

WEB
Scheduler
任务预处理
任务分发
任务结果处理
DB
Redis
节点A
资产发现
漏洞扫描
节点B
资产发现
漏洞扫描
节点C
资产发现
漏洞扫描
节点D
资产发现
漏洞扫描
节点E
资产发现
漏洞扫描
. . . . . .

原理与架构

用户创建任务

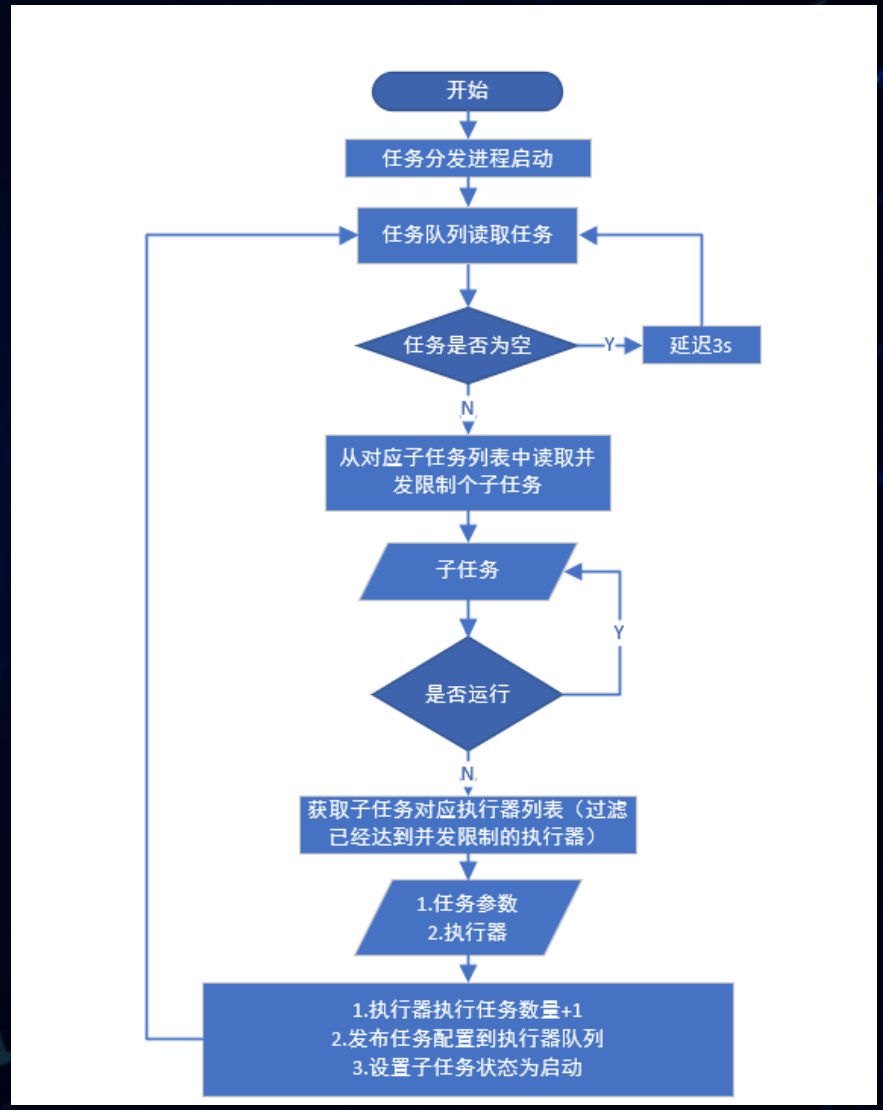
原理与架构

任务预处理


开始
预处理进程启动
预处理队列读取任务
任务是否为空
Y
延迟3s
N
拆分IP段为并发数量*2段
IP段列表
1.存储子任务
2.发布子任务到任务执行队列

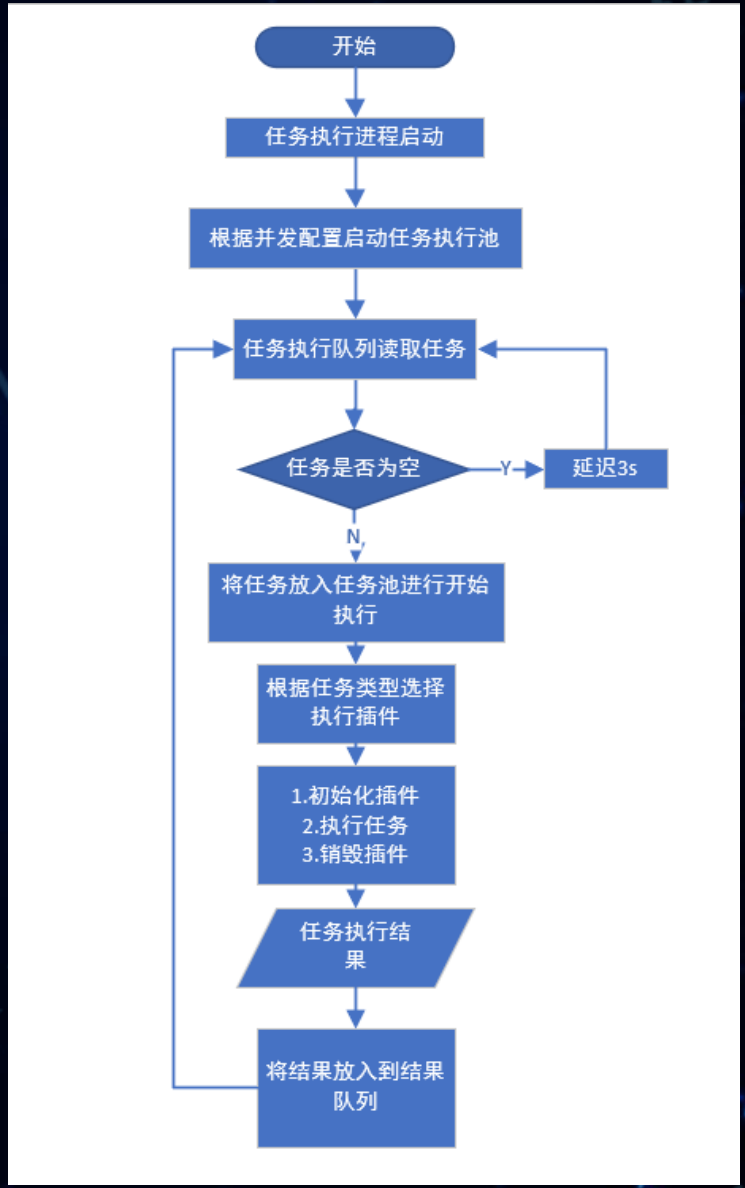
原理与架构

任务分发


开始
任务分发进程启动
任务队列读取任务
任务是否为空
Y
延迟3s
N
从对应子任务列表中读取并发限制个子任务
子任务
Y
是否运行
N
获取子任务对应执行器列表（过滤已经达到并发限制的执行器）
1.任务参数
2.执行器
1.执行器执行任务数量+1
2.发布任务配置到执行器队列
3.设置子任务状态为启动

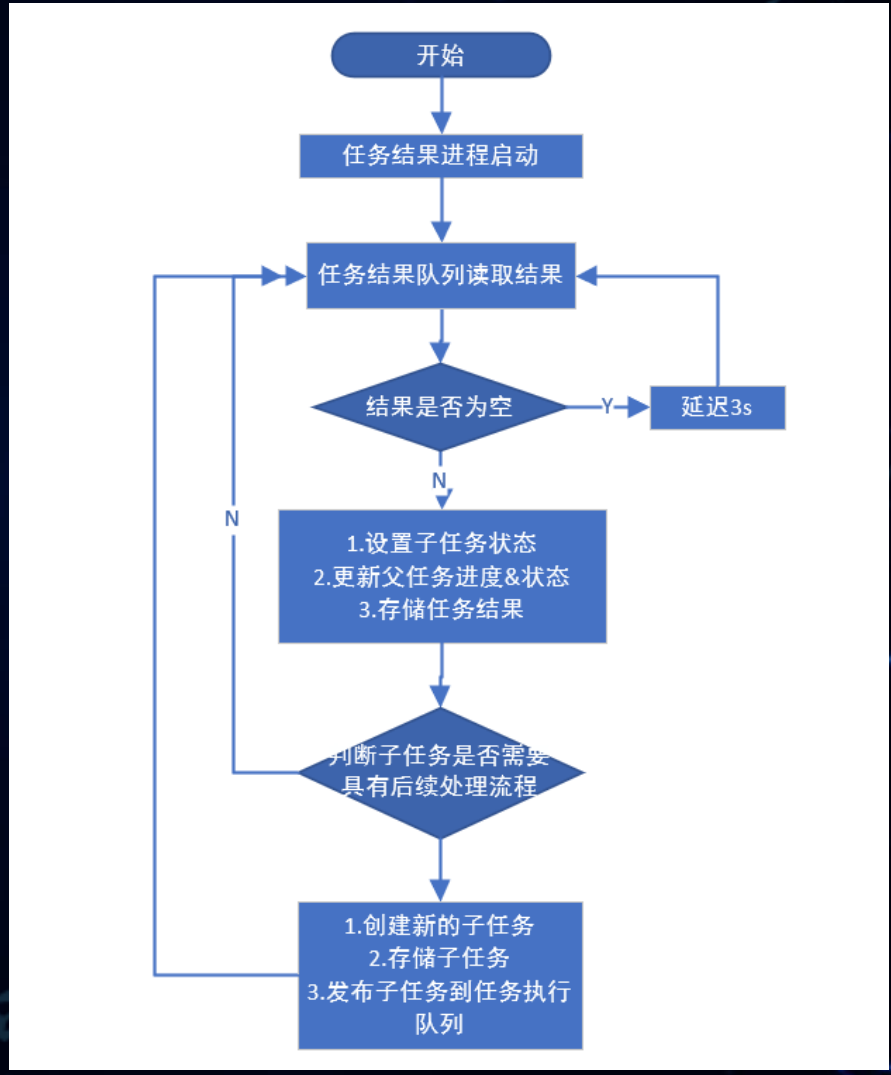
原理与架构

任务执行


开始
任务执行进程启动
根据并发配置启动任务执行池
任务执行队列读取任务
任务是否为空
Y
延迟3s
N
将任务放入任务池进行开始执行
根据任务类型选择执行插件
1.初始化插件
2.执行任务
3.销毁插件
任务执行结果
将结果放入到结果队列

原理与架构

任务结果


开始
任务结果进程启动
任务结果队列读取结果
结果是否为空
Y
延迟3s
N
1.设置子任务状态
2.更新父任务进度&状态
3.存储任务结果
N
具有后续处理流程
1.创建新的子任务
2.存储子任务
3.发布子任务到任务执行队列

原理与架构

其他：

- 插件管理
- 插件参数配置&应用
- 执行器心跳
- 资产扫描
- 资产存储
- 统计

代码实战

IP拆分

```python
class IPUtils(object):

    @staticmethod
    def ip_ranges(ip):
        nodes = []
        for node in ip.replace(',', ' ').replace(';', ' ').split():
            if node.find('-') != -1:
                nodes.append(tuple(map(netaddr.IPAddress, node.split('-'))))
            elif node.find('/') != -1:
                net = netaddr.IPNetwork(node)
                nodes.append([netaddr.IPAddress(net.first), netaddr.IPAddress(net.last)])
            else:
                nodes.append([netaddr.IPAddress(node), netaddr.IPAddress(node)])
        return nodes

    @staticmethod
    def split_ip_ranges(ip, count=3):
        rt = [[] for _ in range(count)]

        for node in IPUtils.ip_ranges(ip):
            start, end = int(node[0]), int(node[1])
            step = math.ceil((end - start + 1) / count)
            idx = 0
            while True:
                tmp = start + step
                if tmp < end:
                    rt[idx].append([netaddr.IPAddress(start), netaddr.IPAddress(tmp)])
                    start += step + 1
                else:
                    rt[idx].append([netaddr.IPAddress(start), netaddr.IPAddress(end)])
                    break

                idx += 1
        return list(filter(lambda x: x, rt))
```

代码实战

Nmap调用扫描

```python
#encoding: utf-8
import time
import logging

import netaddr
import nmap

logger = logging.getLogger(__name__)


def run(job_params, plugin_info, plugin_config, *args, **kwargs):
    hosts = job_params.get('ip', [])
    ports = job_params.get('port', '0-1024')
    hosts = [' '.join(netaddr.iprange_to_globs(*x)) for x in hosts]

    nm = nmap.PortScanner()
    logger.debug(nm.scan(hosts=' '.join(hosts), ports=ports, arguments='-sV -O'))
    for host in nm.all_hosts():
        host_info = nm[host]
        os_info = ''
        try:
            os_info = host_info['osmatch'][0]['name']
        except BaseException as e:
            pass

        apps = []
        for protocol in host_info.all_protocols():
            for port_num, port_info in host_info[protocol].items():
                port = port_info.copy()
                port['protocol'] = protocol
                port['port'] = port_num
                apps.append(port)


        yield host, {
            'ip' : host,
            'name' : host_info.hostname(),
            'apps' : apps,
            'os' : os_info,
            'mac' : host_info.get('addresses').get('mac', ''),
        }
```

代码实战

HTTP请求POC处理

```
def check(ip, plugin_config):
    protocol = plugin_config.get('protocol', 'http')
    port = plugin_config.get('port', DEFAULT_PORTS.get(protocol))
    path = plugin_config.get('path', '/')
    url = '{protocol}://{host}:{port}/{path}'.format(protocol=protocol, host=ip, port=port, path=path.lstrip('/'))
    method = plugin_config.get('method', 'GET').lower()
    cookies = plugin_config.get('cookies', {})
    args = plugin_config.get('args', {})
    body = plugin_config.get('body', {})
    headers = plugin_config.get('headers', {})
    flags = plugin_config.get('flags', {})
    timeout = plugin_config.get('timeout', DEFAULT_TIMEOUT)

    if args:
        url = '{url}?{args}'.format(url, urllib.parse.urlencode(args))

    func = getattr(requests, method, 'get')
    try:
        response = func(url, body, headers=headers, cookies=cookies, timeout=timeout, verify=False)
        check_response = 'check_{0}'.format(flags.get('type', 'status_code'))
        func = globals().get(check_response, None)
        if func is None:
            logger.error('check response func not found: %s', check_response)
            return False, None

        return func(response, flags)
    except BaseException as e:
        logger.exception(e)
        logger.error(traceback.format_exc())
        return False, None
```

代码实战

HTTP请求POC处理

```python
def check_status_code(response, flags):
    status_code = flags.get('status_code', [])
    if not isinstance(status_code, (tuple, list)):
        status_code = [status_code]

    logger.debug('check status code:%s, %s', status_code, response.status_code)
    if response.status_code in status_code:
        return True, {'status_code' : status_code}

    return False, None


def check_header(response, flags):
    header = str(flags.get('header', '')).lower()
    key = str(flags.get('key', '')).lower()
    value = str(flags.get('value', '')).lower()

    logger.debug('check header:%s, %s, %s', header, value, response.headers)
    for k, v in response.headers.items():
        if k.lower() == key:
            return value in v.lower(), {'header' : k, 'value' : v}

    return False, None


def check_response_body(response, flags):
    value = str(flags.get('body', '')).lower()
    text = response.text.lower()
    pos = text.find(value)
    logger.debug('check response_body:%s, %s', value, text)
    if pos != -1:
        length = len(text)
        start = pos - 20
        end = pos + len(value) + 20
        start = 0 if start < 0 else start
        end = length if end > length else end

        return True, {'response' : text[start:end]}

    return False, None
```

代码实战

SOCKET请求POC处理

```python
def check(ip, plugin_config):
    timeout = int(plugin_config.get('timeout', DEFAULT_TIMEOUT))
    port = int(plugin_config.get('port', DEFAULT_PORT))
    text = str(plugin_config.get('request', DEFAULT_TEXT)).encode()
    size = int(plugin_config.get('size', DEFAULT_SIZE))
    flag = str(plugin_config.get('flag', '')).lower()

    try:
        socket.setdefaulttimeout(timeout)
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect((ip, port))
        client.send(text)
        response = client.recv(size)
        response = response.decode().lower()

        pos = response.find(flag)
        if pos != -1:
            length = len(text)
            start = pos - 20
            end = pos + len(value) + 20
            start = 0 if start < 0 else start
            end = length if end > length else end

            return True, {'response' : response[start, end]}
    except BaseException as e:
        logger.exception(e)
        logger.error(e)

    return False, None

```

代码实战

SOCKET请求POC处理

```python
def check(ip, plugin_config):
    timeout = int(plugin_config.get('timeout', DEFAULT_TIMEOUT))
    port = int(plugin_config.get('port', DEFAULT_PORT))
    text = str(plugin_config.get('request', DEFAULT_TEXT)).encode()
    size = int(plugin_config.get('size', DEFAULT_SIZE))
    flag = str(plugin_config.get('flag', '')).lower()

    try:
        socket.setdefaulttimeout(timeout)
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect((ip, port))
        client.send(text)
        response = client.recv(size)
        response = response.decode().lower()

        pos = response.find(flag)
        if pos != -1:
            length = len(text)
            start = pos - 20
            end = pos + len(value) + 20
            start = 0 if start < 0 else start
            end = length if end > length else end

            return True, {'response' : response[start, end]}
    except BaseException as e:
        logger.exception(e)
        logger.error(e)

    return False, None

```

代码实战

SSH弱口令

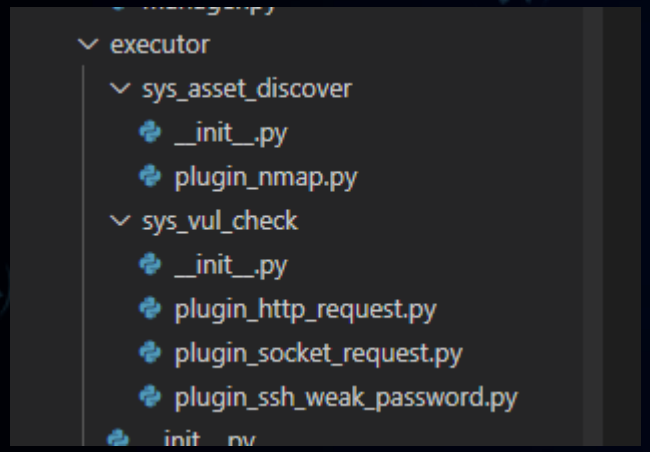
```
def check(ip, ports, usernames, passwords):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    try:
        for port in ports:
            for username in usernames:
                for password in passwords:
                    ssh.connect(ip, port, username, password)
                    return True, {'username' : username, 'password' : password}
    except BaseException as e:
        logger.debug(e)
        logger.debug(traceback.format_exc())
    finally:
        ssh.close()

    return False, None
```

代码实战

插件调度




```python
def init_plugins(self, plugin_pkg):
    plugins = {}
    pkg_name = 'schedule.executor.{0}'.format(plugin_pkg)
    pkg = importlib.import_module(pkg_name)
    paths = getattr(pkg, '__path__', [])
    for path in paths:
        for name in os.listdir(path):
            if name in ('.', '..'):
                continue

            modname, _, suffix = name.rpartition('.')
            if suffix not in ('py', 'pyc', ) or modname in ('__init__'):
                continue

            name = '{0}.{1}'.format(pkg_name, modname)
            module = importlib.import_module(name)
            importlib.reload(module)
            plugins[modname] = module

    return plugins
```

代码实战

插件调度

```python
def run(self, job):
    rt = {}
    job_params = job.get('job_params', {})
    plugins = self.plugins

    logger.debug('executor job: %s, use plugins: %s', job, plugins)

    for name, plugin in plugins.items():
        if self.filter(name, plugin, job):
            logger.debug('plugin not use: %s', name)
            continue

        run = getattr(plugin, 'run', None)
        if run is None:
            logger.error('plugin not found run method: %s', name)
            continue

        getattr(plugin, 'init', NIL)()
        try:
            plugin_info = self.get_plugin_info(name)
            plugin_config = self.get_plugin_config(name)
            logger.debug('run %s plugin, job_params:%s, plugin_info:%s, plugin_config: %s', name, job_params, plugin_info, plugin_config)
            for key, value in run(job_params, plugin_info, plugin_config):
                if value is None:
                    continue
                logger.debug('plugin %s is reback result, [%s]: %s', name, key, value)
                rt.setdefault(key, {'key' : key})
                rt[key].update(value)
        except BaseException as e:
            logger.error(e)
            logger.error(traceback.format_exc())
        finally:
            getattr(plugin, 'destory', NIL)()

    return list(rt.values())
```

Thanks