# agent\nux\portstat.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\

github.com\domeos\agent\nux\portstat.go

## ListeningPorts

**func** ListeningPorts() ([]int64, error) {  
 **return** listeningPorts("ss", "-t", "-l", "-n")  
}

## listeningPorts

**func** listeningPorts(name string, args ...string) ([]int64, error) {  
 ports := []int64{}  
  
 bs, err := sys.CmdOutBytes(name, args...)  
 **if** err != nil {  
 **return** ports, err  
 }  
  
 reader := bufio.NewReader(bytes.NewBuffer(bs))  
  
 // ignore the first line  
 line, err := file.ReadLine(reader)  
 **if** err != nil {  
 **return** ports, err  
 }  
  
 **for** {  
 line, err = file.ReadLine(reader)  
 **if** err == io.EOF {  
 err = nil  
 **break** } **else if** err != nil {  
 **return** ports, err  
 }

// Fields 以连续的空白字符为分隔符，将 s 切分成多个子串，结果中不包含空白字符本身

// 空白字符有：\t, \n, \v, \f, \r, ' ', U+0085 (NEL), U+00A0 (NBSP)

// 如果 s 中只包含空白字符，则返回一个空列表

fields := strings.Fields(string(line))  
 fieldsLen := len(fields)  
  
 **if** fieldsLen != 4 && fieldsLen != 5 {  
 **return** ports, fmt.Errorf("output of %s format not supported", name)  
 }  
  
 portColumnIndex := 2  
 **if** fieldsLen == 5 {  
 portColumnIndex = 3  
 }

//切片和索引s == s[:i]+s[i:]

location := strings.LastIndex(fields[portColumnIndex], ":")  
 port := fields[portColumnIndex][location+1:]

//将s转换为一个整数，strconv.ParseInt(s,base,bits)指定进制的大小（2 ~ 36）

**if** p, e := strconv.ParseInt(port, 10, 64); e != nil {  
 **return** ports, fmt.Errorf("parse port to int64 fail: %s", e.Error())  
 } **else** {  
 ports = append(ports, p)  
 }  
  
 }  
  
 **return** slice.UniqueInt64(ports), nil  
}

# agent\cron\builtin.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src

\github.com\domeos\agent\cron\builtin.go

## syncBuiltinMetrics

**func** syncBuiltinMetrics() {  
  
 **var** timestamp int64 = -1  
 **var** checksum string = "nil"  
  
 duration := time.Duration(g.Config().Heartbeat.Interval) \* time.*Second* **for** {  
 time.Sleep(duration)  
  
 **var** ports = []int64{}  
 **var** paths = []string{}  
 **var** procs = make(**map**[string]**map**[int]string)  
 **var** urls = make(**map**[string]string)  
  
 hostname, err := g.Hostname()  
 **if** err != nil {  
 **continue** }  
  
 req := model.AgentHeartbeatRequest{  
 Hostname: hostname,  
 Checksum: checksum,  
 }  
  
 **var** resp model.BuiltinMetricResponse  
 err = g.HbsClient.Call("Agent.BuiltinMetrics", req, &resp)  
 **if** err != nil {  
 log.Println("ERROR:", err)  
 **continue** }  
  
 **if** resp.Timestamp <= timestamp {  
 **continue** }  
  
 **if** resp.Checksum == checksum {  
 **continue** }  
  
 timestamp = resp.Timestamp  
 checksum = resp.Checksum  
  
 **for** \_, metric := **range** resp.Metrics {  
  
 **if** metric.Metric == g.*URL\_CHECK\_HEALTH* {  
 arr := strings.Split(metric.Tags, ",")  
 **if** len(arr) != 2 {  
 **continue** }  
 url := strings.Split(arr[0], "=")  
 **if** len(url) != 2 {  
 **continue** }  
 stime := strings.Split(arr[1], "=")  
 **if** len(stime) != 2 {  
 **continue** }  
 **if** \_, err := strconv.ParseInt(stime[1], 10, 64); err == nil {  
 urls[url[1]] = stime[1]  
 } **else** {  
 log.Println("metric ParseInt timeout failed:", err)  
 }  
 }

//策略绑定到HostGroup后开始采集端口

**if** metric.Metric == g.*NET\_PORT\_LISTEN* {  
 arr := strings.Split(metric.Tags, "=")  
 **if** len(arr) != 2 {  
 **continue** }  
  
 **if** port, err := strconv.ParseInt(arr[1], 10, 64); err == nil {  
 ports = append(ports, port)  
 } **else** {  
 log.Println("metrics ParseInt failed:", err)  
 }  
  
 **continue** }  
  
 **if** metric.Metric == g.*DU\_BS* {  
 arr := strings.Split(metric.Tags, "=")  
 **if** len(arr) != 2 {  
 **continue** }  
  
 paths = append(paths, strings.TrimSpace(arr[1]))  
 **continue** }  
  
 **if** metric.Metric == g.*PROC\_NUM* {  
 arr := strings.Split(metric.Tags, ",")  
  
 tmpMap := make(**map**[int]string)  
  
 **for** i := 0; i < len(arr); i++ {  
 **if** strings.HasPrefix(arr[i], "name=") {  
 tmpMap[1] = strings.TrimSpace(arr[i][5:])  
 } **else if** strings.HasPrefix(arr[i], "cmdline=") {  
 tmpMap[2] = strings.TrimSpace(arr[i][8:])  
 }  
 }  
  
 procs[metric.Tags] = tmpMap  
 }  
 }  
  
 g.SetReportUrls(urls)  
 g.SetReportPorts(ports)

//设置需要监听的端口

g.SetReportProcs(procs)  
 g.SetDuPaths(paths)  
  
 }  
}

# hbs\rpc\agent.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\

hbs\rpc\agent.go

## BuiltinMetrics

// agent按照server端的配置，按需采集的metric，比如net.port.listen port=22 或者 proc.num name=zabbix\_agentd  
**func** (t \*Agent) BuiltinMetrics(args \*model.AgentHeartbeatRequest, reply \*model.BuiltinMetricResponse) error {  
 **if** args.Hostname == "" {  
 **return** nil  
 }  
  
 metrics, err := cache.GetBuiltinMetrics(args.Hostname)  
 **if** err != nil {  
 **return** nil  
 }  
  
 checksum := ""  
 **if** len(metrics) > 0 {  
 checksum = DigestBuiltinMetrics(metrics)  
 }  
  
 **if** args.Checksum == checksum {  
 reply.Metrics = []\*model.BuiltinMetric{}  
 } **else** {  
 reply.Metrics = metrics  
 }  
 reply.Checksum = checksum  
 reply.Timestamp = time.Now().Unix()  
  
 **return** nil  
}

# hbs\cache\strategies.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\

src\github.com\open-falcon\hbs\cache\strategies.go

## GetBuiltinMetrics

**func** GetBuiltinMetrics(hostname string) ([]\*model.BuiltinMetric, error) {  
 ret := []\*model.BuiltinMetric{}  
 hid, exists := HostMap.GetID(hostname)  
 **if** !exists {  
 **return** ret, nil  
 }  
  
 gids, exists := HostGroupsMap.GetGroupIds(hid)  
 **if** !exists {  
 **return** ret, nil  
 }  
  
 // 根据gids，获取绑定的所有tids  
 tidSet := set.NewIntSet()  
 **for** \_, gid := **range** gids {  
 tids, exists := GroupTemplates.GetTemplateIds(gid)  
 **if** !exists {  
 **continue** }  
  
 **for** \_, tid := **range** tids {  
 tidSet.Add(tid)  
 }  
 }  
  
 tidSlice := tidSet.ToSlice()  
 **if** len(tidSlice) == 0 {  
 **return** ret, nil  
 }  
  
 // 继续寻找这些tid的ParentId  
 allTpls := TemplateCache.GetMap()  
 **for** \_, tid := **range** tidSlice {  
 pids := ParentIds(allTpls, tid)  
 **for** \_, pid := **range** pids {  
 tidSet.Add(pid)  
 }  
 }  
  
 // 终于得到了最终的tid列表  
 tidSlice = tidSet.ToSlice()  
  
 // 把tid列表用逗号拼接在一起  
 count := len(tidSlice)  
 tidStrArr := make([]string, count)  
 **for** i := 0; i < count; i++ {  
 tidStrArr[i] = strconv.Itoa(tidSlice[i])  
 }  
  
 **return** db.QueryBuiltinMetrics(strings.Join(tidStrArr, ","))  
}

# 进程端口监控 | Open-Falcon

https://book.open-falcon.org/zh/usage/proc-port-monitor.html

## 引言

我们说falcon-agent是无需配置即可自动化采集200多项监控指标数据，比如cpu相关的、内存相关的、磁盘io相关的、网卡相关的等等，都可以自动发现，自动采集。

## 端口监控

falcon-agent编写初期是把本机监听的所有端口上报给server端，比如机器监听了80、443、22三个端口，就会自动上报三条数据：

net.port.listen/port=22

net.port.listen/port=80

net.port.listen/port=443

上报的端口数据，value是1，如果后来某些端口不再监听了，那就会停止上报数据。这样是否OK呢？存在两个问题：

* 机器监听的端口可能很多很多，但是真正想做监控的端口可能不多，这会造成资源浪费
* 目前Open-Falcon还不支持nodata监控，端口挂了不上报数据了，没有nodata机制，是发现不了的

改进之。

agent到底要采集哪些端口是通过用户配置的策略自动计算得出的。因为无论如何，监控配置策略是少不了的。比如用户配置了2个端口：

net.port.listen/port=8080 if all(#3) == 0 then alarm()

net.port.listen/port=8081 if all(#3) == 0 then alarm()

将策略绑定到某个HostGroup，那么这个HostGroup下的机器就要去采集8080和8081这俩端口的情况了。这个信息是通过agent和hbs的心跳机制下发的。

agent通过ss -tln拿到当前有哪些端口在监听，如果8080在监听，就设置value=1，汇报给transfer，如果发现8081没在监听，就设置value=0，汇报给transfer。

## 进程监控

进程监控和端口监控类似，也是通过用户配置的策略自动计算出来要采集哪个进程的信息然后上报。举个例子：

proc.num/name=ntpd if all(#2) == 0 then alarm()

proc.num/name=crond if all(#2) == 0 then alarm()

proc.num/cmdline=uic.properties if all(#2) == 0 then alarm()

proc.num表示进程数，比如进程名叫做crond的进程，其实可以有多个。支持两种tag配置，一个是进程name，一个是配置进程cmdline，但是不能同时出现。

那现在DEV写了一个程序，我怎么知道进程名呢？ 首先要拿到进程ID，然后cat /proc/$pid/status，看到里面的name字段了么？falcon-agent就是根据这个name字段来采集的。此处有个坑，就是这个name字段最多15个字节，所以，如果你的进程名特别长可能被截断，截断之前的原始进程名我们不管，agent以这个status文件中的name为准。所以，你配置name这个tag的时候，一定要看一眼这个status文件，从这里获取name，而不是想当然的去写一个你自认为对的进程名。

再说说cmdline，name是从/proc/$pid/status文件采集的，cmdline是从/proc/$pid/cmdline采集的。这个文件存放的是你启动进程的时候用到的命令，比如你用java -c uic.properties启动了一个Java进程，进程名是java，其实所有的java进程，进程名都是java，那我们是没法通过name字段做区分的。怎么办呢？此时就要求助于这个/proc/$pid/cmdline文件的内容了。

cmdline中的内容是你的启动命令，这么说不准确，你会发现空格都没了。其实是把空格自动替换成\0了。不用关心，直接鼠标选中，拷贝之即可。不要自以为是的手工加空格配置到策略中哈，监控策略的tag是不允许有空格的。

上面的例子，java -c uic.properties在cmdline中的内容会变成：java-cuic.properties，无需把整个cmdline都拷贝并配置到策略中。虽然name这个tag是全匹配的，即用的==比较name，但是cmdline不是，我们只需要拷贝cmdline的一部分字符串，能够与其他进程区分开即可。比如上面的配置：

proc.num/cmdline=uic.properties if all(#2) == 0 then alarm()

就已经OK了。falcon-agent拿到cmdline文件的内容之后会使用strings.Contains()方法来做判断

听起来是不是挺复杂的？呵呵，如果你的进程有端口在监听，就配置一个端口监控就可以了，无需既配置端口监控、又配置进程监控，毕竟如果进程挂了，端口肯定就不会监听了。

# <http://192.168.0.179:5050/>

admin/123456

# 清理进程

#清理进程

delete from falcon\_portal.grp where grp.grp\_type=200;

delete from falcon\_portal.tpl where tpl.type=200;

delete from falcon\_portal.domain\_ip;

delete from falcon\_portal.team where team.type=200;

# ln

ln -s /rootfs/sbin/ss /sbin/ss

# yum - update ss utility on centos - Unix & Linux Stack Exchange

http://unix.stackexchange.com/questions/148924/update-ss-utility-on-centos

yum install -y iproute

ss is not in a package called ss. If you run:

yum provides \*/ss

you will be shown a list of packages that provide the ss utility. A quick read through this will show that the package that provides ss is iproute.

You could possibly use a website such as <http://rpmfind.net/linux/RPM/index.html> to search for later versions of iptroute and install that.

The problem with this approach is that later packages will have dependencies that you don't have installed on CentOS 6. You will therefore need to install those too. After a while you'll end up with a hotchpotch system with various versions of utilities and libraries that haven't been thoroughly tested with each other which will only give you a great deal of pain, especially when you attempt an yum update later.

An alternative is to compile it yourself. There are various resources available on the internet. You can either compile and install, or you can compile and create an RPM which you then install on your system. However, compiling will necessitate the installation of many utilities and libraries on your system if you haven't compiled before.

Another option would be to use [Copr build system](https://copr.fedoraproject.org/) to compile remotely for you. You create a src.rpmfor iptables and send it to copr for building. Creating a src.rpm is relatively easy. Even easier, is editing already created src.rpm files from other systems, such as CentOS 7 or Feodra 20. Information on src.rpm is available from the [Fedora Wiki](https://fedoraproject.org/wiki/How_to_create_an_RPM_package), [Fedora Docs](http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch11s03.html) and [CentOS site](http://wiki.centos.org/HowTos/RebuildSRPM) and, of course, on Google.

Edit the .spec file within the src.rpm to reflect the version of the package you want to install, remembering that a certain version of a package (such as iproute) may well need certain version of other libraries installed. In this case, you'll end up in the same situation as if you'd downloaded the rpm from RPM Find.net above. However, using this method gives you slightly more control.