

CDK关键源码分析

- [基本信息探测](#)
- [Cgroup检测](#)
- [ASLR检测分析](#)
- [查询可用命令](#)
- [查询Capabilities](#)
- [mount 信息收集](#)
- [Cloud服务商信息获取](#)
- [敏感环境变量](#)
- [敏感文件路径](#)
- [查询敏感服务](#)

基本信息探测

```
1 func BasicSysInfo() {
2     // current dir(pwd)
3     dir, err := os.Getwd()
4     log.Println("current dir:", dir)
5
6     // current user(id)
7     u, err := user.Current()
8     log.Println("current user:", u.Username, "uid:", u.Uid, "gid:", u.Gid,
9         "home:", u.HomeDir)
10
11     // hostname
12     hostname, err := os.Hostname()
13     log.Println("hostname:", hostname)
14
15     // os/kernel version
16     kversion, _ := host.KernelVersion()
17     platform, family, osversion, _ := host.PlatformInformation()
18     log.Println(family, platform, osversion, "kernel:", kversion)
19 }
```

基本信息探测中主要有三块，包括当前目录、当前用户、主机名、系统内核版本，这些都是直接借助OS、user、Host等package直接可以得到。

Cgroup检测

```
1 var MainPIDCgroup = "/proc/1/cgroup"
2 func DumpMainCgroup() {
```

```

3
4     data, err := ioutil.ReadFile(MainPIDCgroup)
5     scanner := bufio.NewScanner(strings.NewReader(string(data)))
6     for scanner.Scan() {
7         fmt.Printf("\t%s\n", scanner.Text())
8     }
9 }

```

比较直白，直接读取/proc/1/cgroup

ASLR检测分析

```

1 func ASLR() {
2     // ASLR off: /proc/sys/kernel/randomize_va_space = 0
3     var ASLRSetting = "/proc/sys/kernel/randomize_va_space"
4
5     data, err := ioutil.ReadFile(ASLRSetting)
6     if err != nil {
7         log.Printf("err found while open %s: %v\n", RouteLocalNetProcPath,
err)
8         return
9     }
10    log.Printf("/proc/sys/kernel/randomize_va_space file content: %s",
string(data))
11
12    if string(data) == "0" {
13        log.Println("ASLR is disabled.")
14    } else {
15        log.Println("ASLR is enabled.")
16    }
17
18 }
19

```

主要是看/proc/sys/kernel/randomize_va_space的值，如果是0，表示关闭ASLR否则为开启。

查询可用命令

```

1 func SearchAvailableCommands() {
2     ans := []string{}
3     for _, cmd := range conf.LinuxCommandChecklist {
4         _, err := exec.LookPath(cmd)
5         if err == nil {
6             ans = append(ans, cmd)
7         }
8     }
9     log.Printf("available commands:\n\t%s\n", strings.Join(ans, ","))
10 }

```

结合conf目录下LinuxCommandChecklist的定义：

```
var LinuxCommandChecklist = []string{
    "curl", "wget", "nc", "netcat", "kubect1", "docker", "find", "ps", "java", "python", "python3",
    "php", "node", "npm", "apt", "yum", "dpkg", "nginx", "httpd", "apache", "apache2", "ssh", "mysql",
    "mysql-client", "git", "svn", "vi", "capsh", "mount", "fdisk", "gcc", "g++", "make", "base64",
    "python2", "python2.7", "perl", "xterm", "sudo", "ruby",
}
```

可知，CDK是通过遍历LinuxCommandChecklist中的命令并以此执行，利用 `exec.LookPath(cmd)` 执行，如果可以执行就说明可用。

查询Capabilities

```
func GetProcCapabilities() bool {
    data, err := ioutil.ReadFile("/proc/self/status")
    scanner := bufio.NewScanner(strings.NewReader(string(data)))
    log.Println("Capabilities hex of Caps(CapInh|CapPrm|CapEff|CapBnd|CapAmb):")

    for scanner.Scan() {
        line := scanner.Text()
        if strings.HasPrefix(line, "Cap") {
            fmt.Printf("\t%s\n", line)
        }
    }
}

func getAddCaps(currentCaps []string) []string {
    var addCaps []string
    for _, c := range currentCaps {
        if !util.StringContains(capability.DockerDefaultCaps, c) {
            addCaps = append(addCaps, c)
        }
    }
    return addCaps
}
```

这里是通过提取 `/proc/self/status` 下的内容来查询。

执行一下

```
root@ubuntu:~/mytest# go run linux_capabilities.go
2022/03/15 15:12:43 Capabilities hex of Caps(CapInh|CapPrm|CapEff|CapBnd|CapAmb):
CapInh: 0000000000000000
CapPrm: 0000003fffffffff
CapEff: 0000003fffffffff
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
Cap decode: 0x0000003fffffffff = CAP_CHOWN,CAP_DAC_OVERRIDE,CAP_DAC_READ_SEARCH,CAP_FC
Add capability list: CAP_DAC_READ_SEARCH,CAP_LINUX_IMMUTABLE,CAP_NET_BROADCAST,CAP_NET
```

```
[*] Maybe you can exploit the Capabilities below:  
[!] CAP_DAC_READ_SEARCH enabled. You can read files from host. Use 'cdk run cap-dac-read-search'  
[!] CAP_SYS_MODULE enabled. You can escape the container via loading kernel module. More info  
Critical - SYS_ADMIN Capability Found. Try 'cdk run rewrite-cgroup-devices/mount-cgroup/...'.  
Critical - Possible Privileged Container Found.
```

实际上执行: `grep Cap /proc/$BASHPID/status`即可得到

```
root@ubuntu:~# grep Cap /proc/$BASHPID/status  
CapInh: 0000000000000000  
CapPrm: 0000003fffffffff  
CapEff: 0000003fffffffff  
CapBnd: 0000003fffffffff  
CapAmb: 0000000000000000
```

mount 信息收集

```
1 func GetMounts() ([]Mount, error) {  
2     readPath := "/proc/self/mounts"  
3     file, err := os.Open(readPath)  
4     if err != nil {  
5         log.Printf("[Err] Open %s failed.", readPath)  
6         return nil, err  
7     }  
8     defer checkClose(file)  
9     mounts := []Mount(nil)  
10    reader := bufio.NewReaderSize(file, 64*1024)  
11    for {  
12        line, isPrefix, err := reader.ReadLine()  
13        if err != nil {  
14            if err == io.EOF {  
15                return mounts, nil  
16            }  
17            return nil, err  
18        }  
19        if isPrefix {  
20            return nil, syscall.EIO  
21        }  
22        parts := strings.SplitN(string(line), " ", 5)  
23        if len(parts) != 5 {  
24            return nil, syscall.EIO  
25        }  
26        mounts = append(mounts, Mount{parts[0], parts[1], parts[2],  
27            parts[3]})  
    }
```

Cloud服务商信息获取

```
1 func CheckCloudMetadataAPI() {
2     for _, apiInstance := range conf.CloudAPI {
3         cli := goz.NewClient(goz.Options{
4             Timeout: 1,
5         })
6         resp, err := cli.Get(apiInstance.API)
7         if err != nil {
8             log.Printf("failed to dial %s API.",
9 apiInstance.CloudProvider)
10             continue
11         }
12         r, _ := resp.GetBody()
13         if strings.Contains(r.String(), apiInstance.ResponseMatch) {
14             fmt.Printf("\t%s Metadata API available in %s\n",
15 apiInstance.CloudProvider, apiInstance.API)
16             fmt.Printf("\tDocs: %s\n", apiInstance.DocURL)
17         } else {
18             log.Printf("failed to dial %s API.",
19 apiInstance.CloudProvider)
20         }
21     }
22 }
```

这里从配置文件里面读取Cloud API，包括

```
1 var CloudAPI = []cloudAPIS{
2     {
3         CloudProvider: "Alibaba Cloud",
4         API:           "http://100.100.100.200/latest/meta-data/",
5         ResponseMatch: "instance-id",
6         DocURL:         "https://help.aliyun.com/knowledge_detail/49122.html",
7     },
8     {
9         CloudProvider: "Azure",
10        API:           "http://169.254.169.254/metadata/instance",
11        ResponseMatch: "azEnvironment",
12        DocURL:         "https://docs.microsoft.com/en-us/azure/virtual-
13machines/windows/instance-metadata-service",
14    },
15    {
16        CloudProvider: "Google Cloud",
17        API:           "http://metadata.google.internal/computeMetadata/v1/instance/disks/?recursive=true",
18        ResponseMatch: "deviceName",
19        DocURL:         "https://cloud.google.com/compute/docs/storing-
20retrieving-metadata",
21    },
22    {
23        CloudProvider: "Tencent Cloud",
24    },
25 }
```

```

22 | API: "http://metadata.tencentyun.com/latest/meta-data/",
23 | ResponseMatch: "instance-name",
24 | DocURL: "https://cloud.tencent.com/document/product/213/4934",
25 | },
26 | }

```

敏感环境变量

```

1 | func SearchSensitiveEnv() {
2 |     for _, env := range os.Environ() {
3 |         ans, err := regexp.MatchString(conf.SensitiveEnvRegex, env)
4 |         if err != nil {
5 |             log.Println(err)
6 |         } else if ans {
7 |             log.Printf("sensitive env found:\n\t%s", env)
8 |         }
9 |     }
10 | }

```

这段代码中提到的SensitiveEnvRegex定义如下：

```
var SensitiveEnvRegex = "(?i)\\bssh_|k8s|kubernetes|docker|GOPATH"
```

实际上就是在环境变量中查看是否包含有这些敏感字段。

敏感文件路径

```

func SearchLocalFilePath() {

    filepath.Walk(conf.SensitiveFileConf.StartDir, func(path string, info os.FileInfo, err
        for _, name := range conf.SensitiveFileConf.NameList {
            currentPath := strings.ToLower(path)
            //if util.IsSoftLink(currentPath) && util.IsDir(currentPath) {
            //    fmt.Println("skip", currentPath)
            //    return filepath.SkipDir // skip soft link or it will run into
            //}
            if strings.Contains(currentPath, name) {
                fmt.Printf("\t%s - %s\n", name, path)
                if util.IsDir(currentPath) {
                    return filepath.SkipDir // stop dive if sensitive dir
                }
                return nil
            }
        }
    })

    return nil
}

```

这段代码中提到的SensitiveFileConf.NameList包含以下内容：

```
1      StartDir: "/",
2      NameList: []string{
3          `/docker.sock`,    // docker socket (http)
4          `/containerd.sock`, // containerd socket (grpc)
5          `/containerd/s/`,   // containerd-shim socket (grpc)
6          `.kube/`,
7          `.git/`,
8          `.svn/`,
9          `.pip/`,
10         `/.bash_history`,
11         `/.bash_profile`,
12         `/.bashrc`,
13         `/.ssh/`,
14         `.token`,
15         `/serviceaccount`,
16         `.dockerenv`,
17         `/config.json`,
18     },
```

这里是从/开始遍历文件，看是否存在在敏感文件路径列表SensitiveFileConf.NameList中的文件名

查询敏感服务

```
1  func SearchSensitiveService() {
2      processList, err := gops.Processes() //得到进程列表，借助外部package实现
3      if err != nil {
4          log.Println("ps.Processes() Failed, are you using windows?")
5      }
6      for _, proc := range processList {
7          ans, err := regexp.MatchString(conf.SensitiveProcessRegex,
8              proc.Executable())
9          if err != nil {
10             log.Println(err)
11         } else if ans {
12             log.Printf("service found in process:\n\t%d\t%d\t%s\n",
13                 proc.Pid(), proc.PPid(), proc.Executable())
14         }
15     }
16 }
```

这里敏感服务的定义如下：

```
var SensitiveProcessRegex = "(?i)ssh|ftp|http|tomcat|nginx|engine|php|java|python|perl|ruby|ku
```

这里直接借用了gops包来实现进程遍历获取进程列表，实际还是读取/proc文件夹。然后对进程列表中的进程进行遍历，看其是否在敏感服务列表中。