CDK关键源码分析

基本信息探测

```go
func BasicSysInfo() {
	// current dir(pwd)
	dir, err := os.Getwd()
	log.Println("current dir:", dir)

	// current user(id)
	u, err := user.Current()
	log.Println("current user:", u.Username, "uid:", u.Uid, "gid:", u.Gid, "home:", u.HomeDir)

	// hostname
	hostname, err := os.Hostname()
	log.Println("hostname:", hostname)

	// os/kernel version
	kversion, _ := host.KernelVersion()
	platform, family, osversion, _ := host.PlatformInformation()
	log.Println(family, platform, osversion, "kernel:", kversion)

}
```

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

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

	data, err := ioutil.ReadFile(MainPIDCgroup)
	scanner := bufio.NewScanner(strings.NewReader(string(data)))
	for scanner.Scan() {
		fmt.Printf("\t%s\n", scanner.Text())
	}
}
```

比较直白，直接读取/proc/1/cgroup

ASLR检测分析

```
func ASLR() {
	// ASLR off: /proc/sys/kernel/randomize_va_space = 0
	var ASLRSetting = "/proc/sys/kernel/randomize_va_space"

	data, err := ioutil.ReadFile(ASLRSetting)
	if err != nil {
		log.Printf("err found while open %s: %v\n", RouteLocalNetProcPath, err)
		return
	}
	log.Printf("/proc/sys/kernel/randomize_va_space file content: %s", string(data))

	if string(data) == "0" {
		log.Println("ASLR is disabled.")
	} else {
		log.Println("ASLR is enabled.")
	}

}

```

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

```
func SearchAvailableCommands() {
	ans := []string{}
	for _, cmd := range conf.LinuxCommandChecklist {
		_, err := exec.LookPath(cmd)
		if err == nil {
			ans = append(ans, cmd)
		}
	}
	log.Printf("available commands:\n\t%s\n", strings.Join(ans, ","))
}
```

结合conf目录下LinuxCommandChecklist的定义：

```
var LinuxCommandChecklist = []string{
    "curl", "wget","nc","netcat","kubectl","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-searc
[!] 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

```go
func GetMounts() ([]Mount, error) {
        readPath := "/proc/self/mounts"
        file, err := os.Open(readPath)
        if err != nil {
                log.Printf("[Err] Open %s failed.", readPath)
                return nil, err
        }
        defer checkClose(file)
        mounts := []Mount(nil)
        reader := bufio.NewReaderSize(file, 64*1024)
        for {
                line, isPrefix, err := reader.ReadLine()
                if err != nil {
                        if err == io.EOF {
                                return mounts, nil
                        }
                        return nil, err
                }
                if isPrefix {
                        return nil, syscall.EIO
                }
                parts := strings.SplitN(string(line), " ", 5)
                if len(parts) != 5 {
                        return nil, syscall.EIO
                }
                mounts = append(mounts, Mount{parts[0], parts[1], parts[2],
parts[3]})
        }
```

Cloud服务商信息获取

```go
func CheckCloudMetadataAPI() {
	for _, apiInstance := range conf.CloudAPI {
		cli := goz.NewClient(goz.Options{
			Timeout: 1,
		})
		resp, err := cli.Get(apiInstance.API)
		if err != nil {
			log.Printf("failed to dial %s API.", apiInstance.CloudProvider)
			continue
		}
		r, _ := resp.GetBody()
		if strings.Contains(r.String(), apiInstance.ResponseMatch) {
			fmt.Printf("\t%s Metadata API available in %s\n", apiInstance.CloudProvider, apiInstance.API)
			fmt.Printf("\tDocs: %s\n", apiInstance.DocURL)
		} else {
			log.Printf("failed to dial %s API.", apiInstance.CloudProvider)
		}
	}
}
```

这里从配置文件里面读取Cloud API，包括

```go
var CloudAPI = []cloudAPIS{
	{
		CloudProvider: "Alibaba Cloud",
		API:           "http://100.100.100.200/latest/meta-data/",
		ResponseMatch: "instance-id",
		DocURL:        "https://help.aliyun.com/knowledge_detail/49122.html",
	},
	{
		CloudProvider: "Azure",
		API:           "http://169.254.169.254/metadata/instance",
		ResponseMatch: "azEnvironment",
		DocURL:        "https://docs.microsoft.com/en-us/azure/virtual-machines/windows/instance-metadata-service",
	},
	{
		CloudProvider: "Google Cloud",
		API:           "http://metadata.google.internal/computeMetadata/v1/instance/disks/?recursive=true",
		ResponseMatch: "deviceName",
		DocURL:        "https://cloud.google.com/compute/docs/storing-retrieving-metadata",
	},
	{
		CloudProvider: "Tencent Cloud",
                API:           "http://metadata.tencentyun.com/latest/meta-data/",
                ResponseMatch: "instance-name",
                DocURL:        "https://cloud.tencent.com/document/product/213/4934",
        },
}
```

敏感环境变量

```
func SearchSensitiveEnv() {
        for _, env := range os.Environ() {
                ans, err := regexp.MatchString(conf.SensitiveEnvRegex, env)
                if err != nil {
                        log.Println(err)
                } else if ans {
                        log.Printf("sensitive env found:\n\t%s", env)
                }
        }
}
```

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：

```
		StartDir: "/",
		NameList: []string{
			`/docker.sock`,     // docker socket (http)
			`/containerd.sock`, // containerd socket (grpc)
			`/containerd/s/`,   // containerd-shim socket (grpc)
			`.kube/`,
			`.git/`,
			`.svn/`,
			`.pip/`,
			`/.bash_history`,
			`/.bash_profile`,
			`/.bashrc`,
			`/.ssh/`,
			`.token`,
			`/serviceaccount`,
			`.dockerenv`,
			`/config.json`,
		},
```

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

```
func SearchSensitiveService() {
	processList, err := gops.Processes() //得到进程列表，借助外部package实现
	if err != nil {
		log.Println("ps.Processes() Failed, are you using windows?")
	}
	for _, proc := range processList {
		ans, err := regexp.MatchString(conf.SensitiveProcessRegex, 
proc.Executable())
		if err != nil {
			log.Println(err)
		} else if ans {
			log.Printf("service found in process:\n\t%d\t%d\t%s\n", 
proc.Pid(), proc.PPid(), proc.Executable())
		}
	}
}
```

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。