

# Offensive Development

---



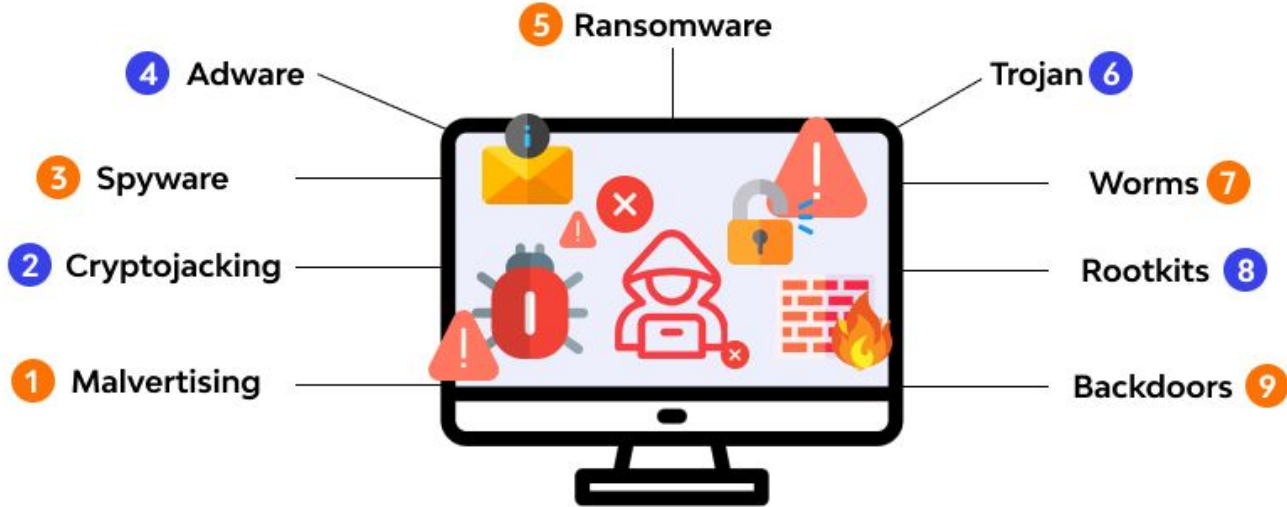
*"Welcome to the dark side"*





Malware - a program developed with the intent to cause harm.

## Types of malware





# Server side (C2 server)

---



Server - You can create a server with any language. Python, nodejs, php ....

We are going to use golang which is effective in managing resources (handles many requests with great speed ).

<https://go.dev/doc/install>

[https://code.visualstudio.com/docs/?dv=linux64\\_deb](https://code.visualstudio.com/docs/?dv=linux64_deb)

Versioning your code is important for reverting and making changes to products that are in production. We will use git to version control our c2.

```
sudo apt install git
git config --global user.email "example@email.com"
git config --global user.name "example"
```



Create a project folder

```
mkdir <projectname>
```

```
cd <projectname>
```

Initialize the git repository using

```
git init
```

Initialize our project as a module that can be used in other projects. Where they are publicly accessible. For our case we don't need to reuse this module hence we can use example.

```
go mod init example/<projectname>
```

Create a program that prints “Welcome”

Commit the changes.

```
git status
```

```
git add <filename>
```

```
git commit -m “welcome “
```

```
git status
```

```
main.go > ...
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Sample running")
7 }
8
```



Ensure you don't have sensitive paths and strings in your code by using the .env file.

Create a branch to add our new feature.

```
git branch  
git branch dev  
git checkout dev
```

Download a module that another person has written.

```
go get "github.com/joho/godotenv"  
go mod tidy
```

Create .env file that has our variables. Create .gitignore file to prevent env file from being published.

Test and if it works merge it to main branch . Then delete the branch.

```
git status  
git add .  
git commit -m "Environment"  
git checkout main  
git merge dev  
git branch -d dev
```





```
4      "fmt"
5      "os"
6
7      "github.com/joho/godotenv"
8  )
9
10 func main() {
11     err := godotenv.Load()
12     if err != nil {
13         fmt.Println("Error loading environment")
14         return
15     }
16     fmt.Println(os.Getenv("server")+" "+os.Getenv("port"))
17 }
18
```

```
1 .*
```

```
1 server=127.0.0.1
2 port=844
```

```
seid@Dark:~/Desktop/Sample$ go get "github.com/joho/godotenv"
go: downloading github.com/joho/godotenv v1.4.0
go: added github.com/joho/godotenv v1.4.0
seid@Dark:~/Desktop/Sample$ go mod tidy
seid@Dark:~/Desktop/Sample$ go run main.go
127.0.0.1:844
seid@Dark:~/Desktop/Sample$ git add .
seid@Dark:~/Desktop/Sample$ git commit -m "environment"
[dev 388a578] environment
3 files changed, 17 insertions(+), 3 deletions(-)
create mode 100644 go.sum
seid@Dark:~/Desktop/Sample$ git checkout master
Switched to branch 'master'
seid@Dark:~/Desktop/Sample$ git merge dev
Updating 1c158e6..388a578
Fast-forward
 go.mod | 2 ++
 go.sum | 2 ++
 main.go | 16 ++++++++
 3 files changed, 17 insertions(+), 3 deletions(-)
 create mode 100644 go.sum
seid@Dark:~/Desktop/Sample$ git branch -d dev
Deleted branch dev (was 388a578).
seid@Dark:~/Desktop/Sample$
```

<https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging>



Setup up a listener.

Secure your connection using https. Try using default ports to masquerade your traffic.

<https://ubuntu.com/server/docs/security-certificates>

```
openssl req -new -newkey rsa:2048 -nodes -keyout server.key -out server.csr
```

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

```
SAMPLE
.env
.gitignore
go.mod
go.sum
main.go
server.crt
server.csr
server.key

main.go
1 package main
2
3 import (
4     "fmt"
5     "net/http"
6     "os"
7
8     "github.com/joho/godotenv"
9 )
10
11 func main() {
12     err := godotenv.Load()
13     if err != nil {
14         fmt.Println("Error loading environment")
15         return
16     }
17     fmt.Println(os.Getenv("server")+os.Getenv("port"))
18     err:=http.ListenAndServeTLS(os.Getenv("port"),os.Getenv("cert"),os.Getenv("privkey"),nil)
19     if err!=nil{
20         fmt.Println(err)
21     }
22 }
```

```
.gitignore
1 .*
2 server*
```

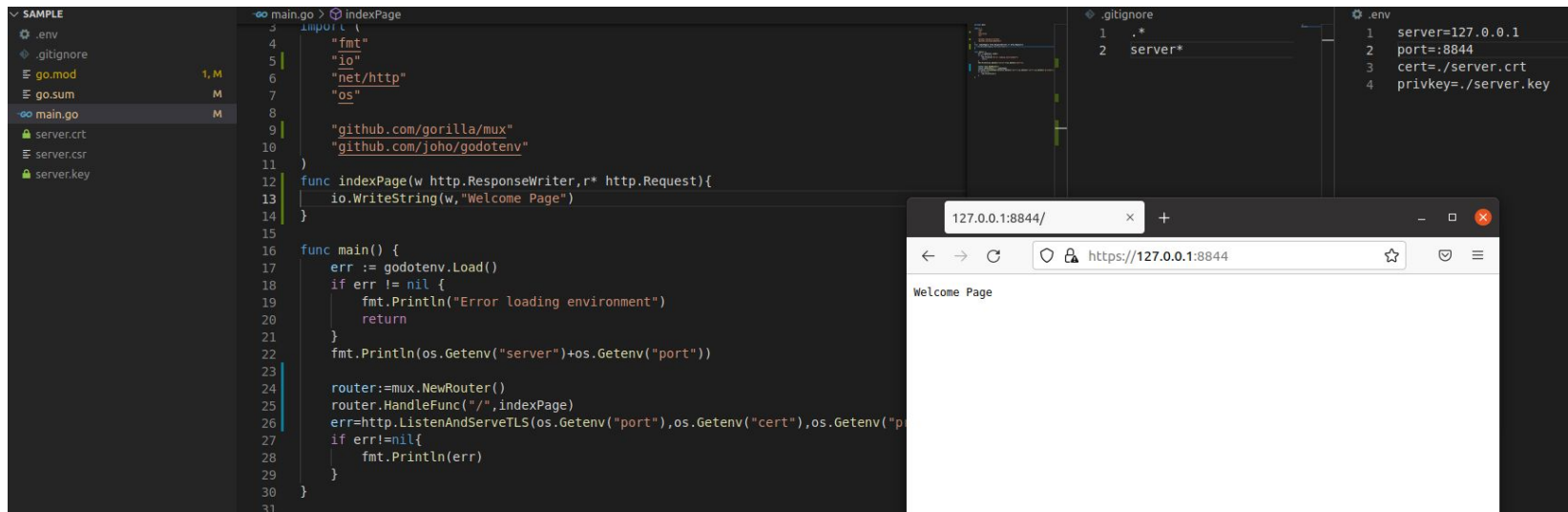
```
.env
1 server=127.0.0.1
2 port=:8844
3 cert=./server.crt
4 privkey=./server.key
```



Setup the Api paths that the agent will communicate to, Use a routing package such as mux.

<https://github.com/gorilla/mux.git>

Make sure they are descriptive and not suspicious.



The screenshot displays a Go development environment with a file explorer on the left, a code editor in the center, and a web browser on the right. The file explorer shows a project structure with files like .env, .gitignore, go.mod, go.sum, main.go, server.crt, server.csr, and server.key. The code editor shows the main.go file with the following code:

```
3  import (
4      "fmt"
5      "io"
6      "net/http"
7      "os"
8
9      "github.com/gorilla/mux"
10     "github.com/joho/godotenv"
11 )
12 func indexPage(w http.ResponseWriter, r * http.Request){
13     io.WriteString(w, "Welcome Page")
14 }
15
16 func main() {
17     err := godotenv.Load()
18     if err != nil {
19         fmt.Println("Error loading environment")
20         return
21     }
22     fmt.Println(os.Getenv("server")+os.Getenv("port"))
23
24     router:=mux.NewRouter()
25     router.HandleFunc("/", indexPage)
26     err:=http.ListenAndServeTLS(os.Getenv("port"), os.Getenv("cert"), os.Getenv("key"), router)
27     if err!=nil{
28         fmt.Println(err)
29     }
30 }
31
```

The web browser on the right shows the URL `https://127.0.0.1:8844` and displays the text "Welcome Page".



Storage of all data in a database. Use a non relations database such as mongodb. You can also cache the results using redis to improve the speed of your queries.

<https://github.com/mongodb/mongo-go-driver>

```
SAMPLE
  Database
  database.go 3, U
  go.mod      U
  go.sum      U
  .env
  .gitignore
  go.mod      M
  go.sum      M
  main.go     M
  server.crt
  server.csr
  server.key

Database > database.go > ...
1 package db
2
3 import (
4     "context"
5     "log"
6     "os"
7     "time"
8
9     "go.mongodb.org/mongo-dr
10    "go.mongodb.org/mongo-dr
11    "go.mongodb.org/mongo-dr
12 )
13
14 type Database struct {
15     Client *mongo.Client
16     Ctx     context.Context
17     Err     error
18 }
19
20 func (db *Database) Connect(
21     db.Client, db.Err = mong
22     if db.Err != nil {
23         log.Fatal(db.Err)
24     }
25     db.Ctx, _ = context.With
26     db.Err = db.Client.Conne
27     if db.Err != nil {

main.go
9      "github.com/gorilla/mux"
10     "github.com/joho/godotenv"
11     "example/db"
12     "go.mongodb.org/mongo-driver/bson"
13     "encoding/json"
14     "strconv"
15 )
16 var database db.Database
17
18 type User struct{
19     Name string;
20     Age int;
21 }
22
23 > func indexPage(w http.ResponseWriter
24 )
25 func getDataHandler(w http.ResponseWriter
26     filter := bson.M{}
27     result, err := database.QueryM
28     var users []User
29     if err == nil {
30         for _, x := range result {
31             doc, _ := bson.Marshal
32             var user User
33             bson.Unmarshal(doc, &u
34             users = append(users, u
35         }
```



You can use a test framework for go lang depending on your preferences (TDD). You can also test out the API using postman.

The screenshot displays the Postman interface for configuring a POST request. The top bar shows a list of recent requests, with the current one being a POST to `https://127.0.0.1:8443`. The main area shows the request details for `https://127.0.0.1:8443/insertData`. The request is configured as a POST method. The body is set to `x-www-form-urlencoded`. The body tab is active, showing a table with two rows of data: `pname` with value `abc` and `page` with value `20`. The bottom of the interface shows a table with columns `KEY`, `VALUE`, and `DESCRIPTION`.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> pname	abc	
<input checked="" type="checkbox"/> page	20	
Key	Value	Description

