[Web] superbee

```
62
63  func (this *BaseController) Prepare() {
64      controllerName, _ := this.GetControllerAndAction()
65      session := this.Ctx.GetCookie(Md5("sess"))
66
67      if controllerName == "MainController" {
68          if session == "" || session != Md5(admin_id + auth_key) {
69              this.Redirect("/login/login", 403)
70              return
71          }
72      } else if controllerName == "LoginController" {
73          if session != "" {
74              this.Ctx.SetCookie(Md5("sess"), "")
75          }
76      } else if controllerName == "AdminController" {
77          domain := this.Ctx.Input.Domain()
78
79          if domain != "localhost" {
80              this.Abort("Not Local")
81              return
82          }
83      }
84  }
85
```

문제의 소스 코드 중 페이지를 라우팅 해주는 메인 코드는 위와 같다
Beego 프레임워크에서 controller 이름과 함수명으로 url path 가 결정된다.
예를 들어 MainController 는 host:port/main/*에서의 동작을 관리한다는 뜻이다.

위의 코드에서 main 에서는 md5(“sess”)이름의 쿠키 값이 if 조건을 통과 못 할 경우 login 으로 리다이렉트한다.

Login 에서는 session 에 값이 있으면 값을 비워버린다 admin 에서는 host 가 localhost 로 설정되지 않았을 경우 “Not Local”이라는 값이 반환되도록 한다.

```
func (this *MainController) Index() {
    this.TplName = "index.html"
    this.Data["app_name"] = app_name
    this.Data["flag"] = flag
    this.Render()
}
```

/main/index 에서 flag 를 볼 수 있다.

```
func (this *LoginController) Login() {
    this.TplName = "login.html"
    this.Data["app_name"] = app_name
    this.Render()
}

func (this *LoginController) Auth() {
    id := this.GetString("id")
    password := this.GetString("password")

    if id == admin_id && password == admin_pw {
        this.Ctx.SetCookie(Md5("sess"), Md5(admin_id + auth_key), 300)

        this.Ctx.WriteString("<script>alert('Login Success');location.href='/main/index';</script>")
        return
    }
    this.Ctx.WriteString("<script>alert('Login Fail');location.href='/login/login';</script>")
}
```

/login/login 에서 login.html 을 렌더링한다. Login.html 에서는 id, pw 를 post 로 입력받고 /login/auth 로 리다이렉트한다.
Login/auth 에서 입력받은 id 아 pw 를 app.conf 에 저장된 값과 비교해서 md5(“sess”)의 쿠키 값을 md5(“admin_id+auth_key)로 설정해준다.

여기서 if 문의 조건식을 우회할 방법을 찾았으나 적절한 방법을 찾지 못 했다. 그래서 다른 방법을 생각하다가 쿠키값을 구해서 직접 세팅한 후 main/index 에 접근하면 되겠다고 생각했다.

```
func (this *AdminController) AuthKey() {
    encrypted_auth_key, _ := AesEncrypt([]byte(auth_key), []byte(auth_crypt_key))
    this.Ctx.WriteString(hex.EncodeToString(encrypted_auth_key))
}
```

/admin/authkey 를 보면 auth_key 가 auth_crypt_key 로 aes 암호화된 값을 반환하는 것을 알 수 있다.
이 부분에 접근해서 암호화된 값을 얻고 그 값으로 쿠키값을 구해서 flag 에 접근하면 되겠다.

```
086c24fea4bf70f45b0080e74d00fb3dcf5ecaad607aeb0c91e9b194d9f9f9e263cebd55cdf1ec2a327d033be657c2582de2ef1ba6d77fd227840116076bdf80
```

프록시툴로 패킷을 잡아서 Host 헤더를 localhost 로 바꿔주면 위와 같이 암호화된 값이 나온다.
이걸 decrypt 하기 위해 key 를 알아야 하는데 주어진 app.conf 파일을 보면 아래처럼 auth_crypt_key 와 관련된 값이 들어있지 않다.

```
app_name = superbee
auth_key = [----------REDEACTED------------]
id = admin
password = [----------REDEACTED------------]
flag = [----------REDEACTED------------]
```

여기서 조금 방황했는데 잘 생각해보니 다른 가려야 하는 값들은
“[----------REDEACTED------------]” 이렇게 변경했는데 key 값은 그냥 없어서 출제자의 의도가 key 값을 실수로 설정 안 한 상황에서의 암호화를 문제에 냈다고 생각했다.

그래서 auth_crypt_key 가 없다고 가정하고 암호화 코드를 분석했다.

```go
func AesEncrypt(origData, key []byte) ([]byte, error) {
    padded_key := Padding(key, 16)
    block, err := aes.NewCipher(padded_key)
    if err != nil {
        return nil, err
    }
    blockSize := block.BlockSize()
    origData = Padding(origData, blockSize)
    blockMode := cipher.NewCBCEncrypter(block, padded_key[:blockSize])
    crypted := make([]byte, len(origData))
    blockMode.CryptBlocks(crypted, origData)
    return crypted, nil
}

func Padding(ciphertext []byte, blockSize int) []byte {
    padding := blockSize - len(ciphertext)%blockSize
    padtext := bytes.Repeat([]byte{byte(padding)}, padding)
    return append(ciphertext, padtext...)
}
```

위 코드를 보면 key 값을 16 바이트만큼 padding 한다. 이후 blocksize 를 구하고 padding 한 값에서 그 사이즈만큼을 Iv 값으로 사용해서 cbc mode 로 암호화한다. Key, padding, block 등 사이즈가 전부 16 바이트이다.

또 key 가 padding 을 거친 후 iv 로 쓰이는데 이 부분에서 key == iv 임을 알 수 있었다.

이를 토대로 찾아보니 위의 암호화 방식은 aes-128/cbc/pkcs7padding 인 것을 알 수 있었고 이에 맞게 복호화 코드를 작성했다.

```python
class AESCryptoCBC():
    def __init__(self, key):
        from Crypto.Cipher import AES
        self.key = key
        iv = key
        self.crypto = AES.new(key, AES.MODE_CBC, iv)

    def decrypt(self, encrypted):
        decrypted = self.crypto.decrypt(encrypted)
        return decrypted

def padding(val):
    pad = 16 - len(val)
    return pad.to_bytes(1, 'little') * pad

key = b'\x10' *16

enc = b'\x00\xfb\x3d\xcf\x5e\xca\xad\x60\x7a\xeb\x0c\x91\xe9\xb1\x94\xd9\xf9\xf9\xe2\x63\xce\xbd\x55\xcd\xf1\xec\x2a\x32\x7d\x03\x3


dec = AESCryptoCBC(key).decrypt(enc)
print("\n", type(dec), dec)
```

Padding 의 경우 항상 input 사이즈가 같게 유지하기 위해 데이터를 제외하고 부족한 바이트를 값으로 해서 나머지 바이트를 채운다. 즉 key 가 설정되지 않았기 때문에 16 바이트를 16 으로 채워서 key 로 쓰면 된다.

그래서 key 를 0x10 16 개로 설정하고 아까 받은 암호문을 바이트단위로 나눠서 저장 후 복호화를 진행했다.

```
<class 'bytes'> b'Th15_sup3r_s3cr3t_K3y_N3v3r_B3_L34k3d\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b'
```

이렇게 auth_key 평문을 구했다.

“Th15_sup3r_s3cr3t_K3y_N3v3r_B3_L34k3d” 이제 이 값을 admin 과 더해서 md5 해싱하고 “sess”를 해싱한 값의 쿠키에 넣어서 /main/index 에 접근하면 된다.

Md5(“sess”) = “f5b338d6bca36d47ee04d93d08c57861”
Md5(“admin_id+auth_key”) = “e52f118374179d24fa20ebcceb95c2af”

```
> document.cookie =
  "f5b338d6bca36d47ee04d93d08c57861=e52f118374179d24fa20ebcceb95c2af"
< 'f5b338d6bca36d47ee04d93d08c57861=e52f118374179d24fa20ebcceb95c2af'
```

Index

codegate2022{d9adbe86f4ecc93944e77183e1dc6342}

Flag : codegate2022{d9adbe86f4ecc93944e77183e1dc6342}