Go Secure Coding Practices
Sulhaedir

IT Security Analyst at Tokopedia

sulhaedir05@gmail.com
• What is secure coding?
• Advantage of secure coding
  → For Pentester
  → For Developer / Programmer
• Are you familiar with Go?
• Common Vulnerabilities in real Go web application
  → sql injection, xss, idor, broken authentication + mitigation
• Best Practices
• Introduction to GoVWA (Go Vulnerable Web Application)
What is Secure Coding?

“What secure coding is the practice of writing programs that are resistant to attack by malicious or mischievous people or programs.”
Advantage Of Secure Coding

• For Pentester

Web Application Exploits

This exploit category includes exploits for web applications.

<table>
<thead>
<tr>
<th>Date</th>
<th>D</th>
<th>A</th>
<th>V</th>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-11-13</td>
<td></td>
<td></td>
<td></td>
<td>Kirby CMS &lt; 2.5.7 - Cross-Site Scripting</td>
<td>Ishaq Mohammed</td>
</tr>
<tr>
<td>2017-11-13</td>
<td></td>
<td></td>
<td></td>
<td>Web Viewer 1.0.0.193 (Samsung SRN-1670D) - Unrestricted File Upload</td>
<td>0xFFFFFFF</td>
</tr>
<tr>
<td>2017-11-07</td>
<td></td>
<td></td>
<td></td>
<td>ManageEngine Applications Manager 13 - SQL Injection</td>
<td>Cody Sixteen</td>
</tr>
<tr>
<td>2017-11-07</td>
<td></td>
<td></td>
<td></td>
<td>pfSense 2.3.1_1 - Command Execution</td>
<td>s4squatch</td>
</tr>
<tr>
<td>2017-11-04</td>
<td></td>
<td></td>
<td></td>
<td>WordPress Plugin Userpro &lt; 4.9.17.1 - Authentication Bypass</td>
<td>Colette...</td>
</tr>
<tr>
<td>2017-11-03</td>
<td></td>
<td></td>
<td></td>
<td>Logitech Media Server 7.9.0 - 'Radio URL' Cross-Site Scripting</td>
<td>Dewank Pant</td>
</tr>
<tr>
<td>2017-11-03</td>
<td></td>
<td></td>
<td></td>
<td>Logitech Media Server 7.9.0 - 'favorites' Cross-Site Scripting</td>
<td>Dewank Pant</td>
</tr>
</tbody>
</table>
For Developer / Programmer
Are You Familiar with Go?

```go
flag.Usage = func() {
    fmt.Fprintf(os.Stderr, "gotags version %s\n\n", VERSION)
    fmt.Fprintf(os.Stderr, "Usage: %s [options] file(s)\n\n", Arg(...))
    flag.PrintDefaults()
}

func main() {
    flag.Parse()

    if printVersion {
        fmt.Fprintf("gotags version %s\n", VERSION)
        return
    }

    if flag.NArg() == 0 {
        // Code here...
    }
```
OWASP Top 10 - 2017

- A1: 2017- Injection
- A2: 2017- Broken Authentication
- A3: 2017- Sensitive Data Exposure
- A4: 2017- XML External Entities (XXE) [NEW]
- A5: 2017- Broken Access Control [Merged]
- A6: 2017- Security Misconfiguration
- A7: 2017- Cross-Site Scripting (XSS)
- A8: 2017- Insecure Deserialization [NEW, Community]
- A9: 2017- Using Components with Known Vulnerabilities
- A10: 2017- Insufficient Logging & Monitoring [NEW, Comm.]
• Sql Injection

```go
id := r.FormValue("id")
sql := fmt.Sprintf(`select * from data where id=%s`, id)
rows, err := db.Query(sql)

http://localhost:8888/getdata?id=1
(select * from data where id=1)

http://localhost:8888/getdata?id=-1+union
select+null,null,database()--

select * from data where id=-1 union select
null,null,database()--
```
• Mitigation using prepare statement

```Kansas
id := r.FormValue("id")

conts sql = `select * from data where id=?`

stmt, err := db.Prepare(sql)

err := stmt.Query(id).Scan(&data)
```
• Cross Site Scripting

```go
func xss(w http.ResponseWriter, r *http.Request) {
    name := r.FormValue("name")
    http://localhost:8888/xss?=tes
    io.WriteString(w,name)
}
```
http://localhost:8888/xss?=<script>alert('tes')</script>
• How about implementing content-type?

```java
w.Header.Set("Content-type":"text/plain")
```
• Use HTML template package for output encoding

name := r.FormValue("name")
template := template.Must(template.ParseGlob("xss.html"))
data["Name"] = name
err := template.ExecuteTemplate(w, name, data)
&lt;script&gt;alert(&#39;tes&#39;)&lt;/script&gt;
• Common Big Mistake

```go
name := r.FormValue("name")

boldName := template.HTML(fmt.Sprintf("<b>%s</b>", name))

template := template.Must(template.ParseGlob("xss.html"))
data = make(map[string]interface{})
data["Name"] = boldName

err := template.ExecuteTemplate(w, name, data)
```
name=<script>alert('xss');</script>
• Insecure Direct Object References (IDOR)

```go
type r struct {
    FormValue func(string) string
}

uid := r.FormValue("uid")
name := r.FormValue("name")

const sql = `update profile set name=? where uid=?`
stmt, _ := db.Prepare(sql)
affected, err := stmt.Exec(name, uid)
```

http://localhost:8888/update?name=jack&id=1

http://localhost:8888/update?name=jack&id=2
• Insecure Direct Object References (IDOR) mitigation

```go
uid := session.GET("uid")
name := r.FormValue("name")

const sql = `update profile set name=? where uid=?`
stmt, _ := db.Prepare(sql)
affected, err := stmt.Exec(name, uid)
```

http://localhost:8888/update?name=jack
• Client side authentication

```plaintext
resp := {}
otp := r.FormValue("otpcode")
uid := session.GET("uid")
lotp := getOTPonDB(uid)
if otp != lotp{
    resp.Status = 0 //failed
}else{
    resp.Status = 1 //success
}
reponseJSON(resp)
```
var data = $('#otpform').serialize()
url = "http://localhost/validate"
$.post(url, data)
.done(function(res){
    if (res.Status === 1){
        document.location.replace("http://localhost:8888/verify")
    } else{
        showErrorAlert()
    }
})
HTTP/1.1 200 OK
Server: nginx
Date: Sun, 12 Nov 2017 05:19:52 GMT
Content-Type: application/json
Connection: close
Vary: Accept-Encoding
Vary: Accept-Encoding
Content-Length: 13

{"Status":0} to{"Status":1}
## Handling User Input

When to use input validation, escaping and safeHTML

<table>
<thead>
<tr>
<th>Require action</th>
<th>Input Validation</th>
<th>Escaping</th>
<th>Safe HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rendered as text</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Input to be added to javascript</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Input to be added as a parameter to a URL</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Rendered as HTML</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Modern XSS Prevention

- Implementing Content Security Policy (CSP)
- Use of Prepared Statements (with Parameterized Queries).
- White List Input
- Escaping All User Supplied Input
- Enforcing Least Privilege
● Centralized authorization routine

● Control access to protect resource

● Authorization matrix (auth check every page)

● Never implement client-side authorization

● Separate application for administrator and user access
Welcome to GoVWA

GoVWA (Go Vulnerable Web Application) is a web application developed to help the pentester and programmers to learn the vulnerabilities that often occur in web applications which is developed using golang. Vulnerabilities that exist in GoVWA are the most common vulnerabilities found in web applications today. So it will help programmers recognize vulnerabilities before they happen to our app. Govwa can also be an additional application of your pentest lab for learning and teaching.

Warning!

Since GoVWA is a web application that contains a vulnerability, never upload govwa to web hosting that can be accessed publicly, because it can cause your server to get hacked. As a suggestion to use GoVWA locally.

https://github.com/0c34/govwa
Thanks.

Any Questions?