Go Secure Coding Practices

OWASP

The Open Web Application Security Project

Introductions

OWASP
The Open Web Application Security Project

Sulhaedir

IT Security Analyst at Tokopedia

sulhaedir05@gmail.com

Outline

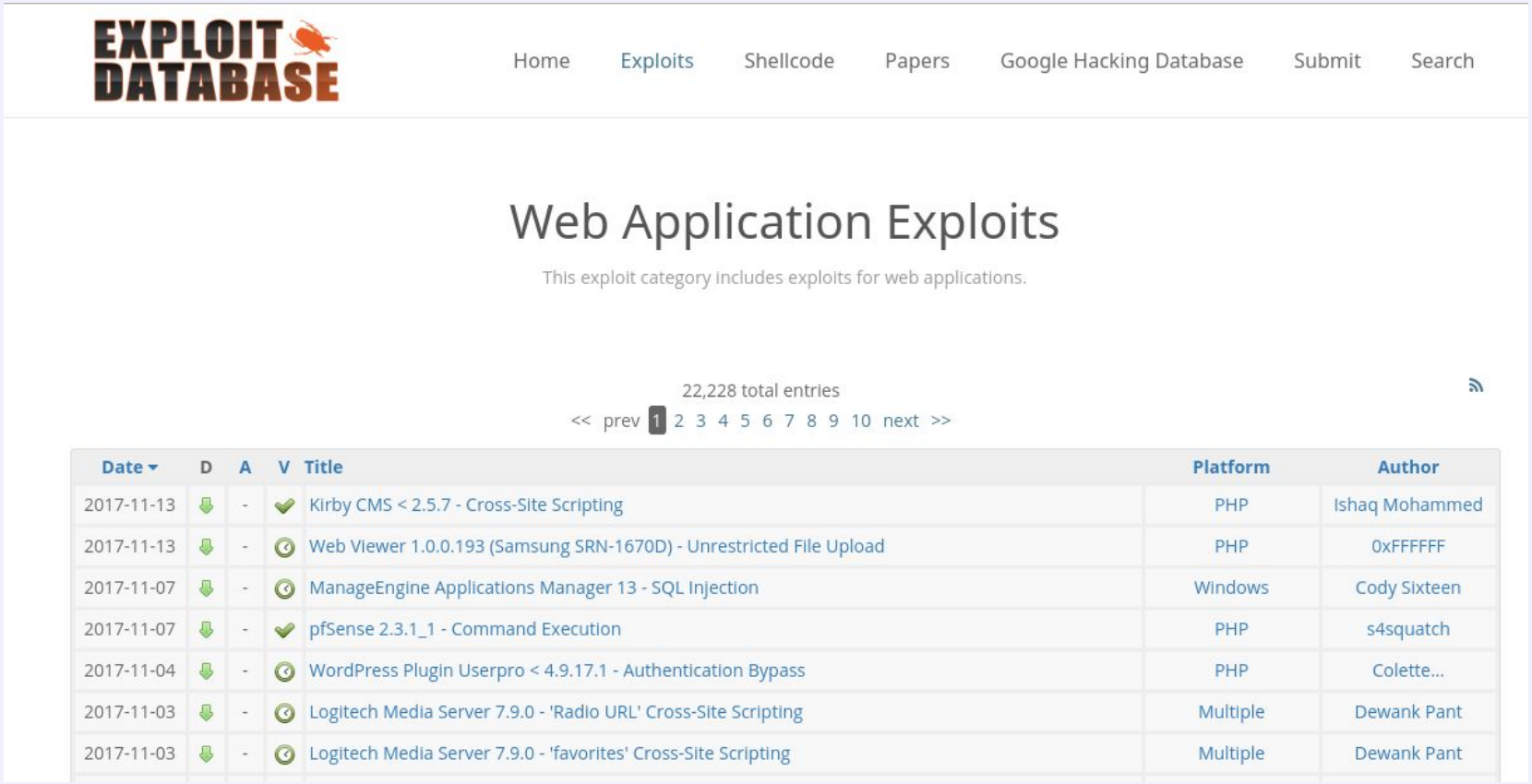
- What is secure coding?
- Advantage of secure coding
  - → For Pentester
  - → For Developer / Programmer
- Are you familiar with Go?
- Common Vulnerabilities in real Go web application
  - → sql injection, xss, idor, broken authentication + mitigation
- Best Practices
- Introduction to GoVWA (Go Vulnerable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

What is Secure Coding?

"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



EXPLOIT DATABASE

Home Exploits Shellcode Papers Google Hacking Database Submit Search

Web Application Exploits

This exploit category includes exploits for web applications.

22,228 total entries

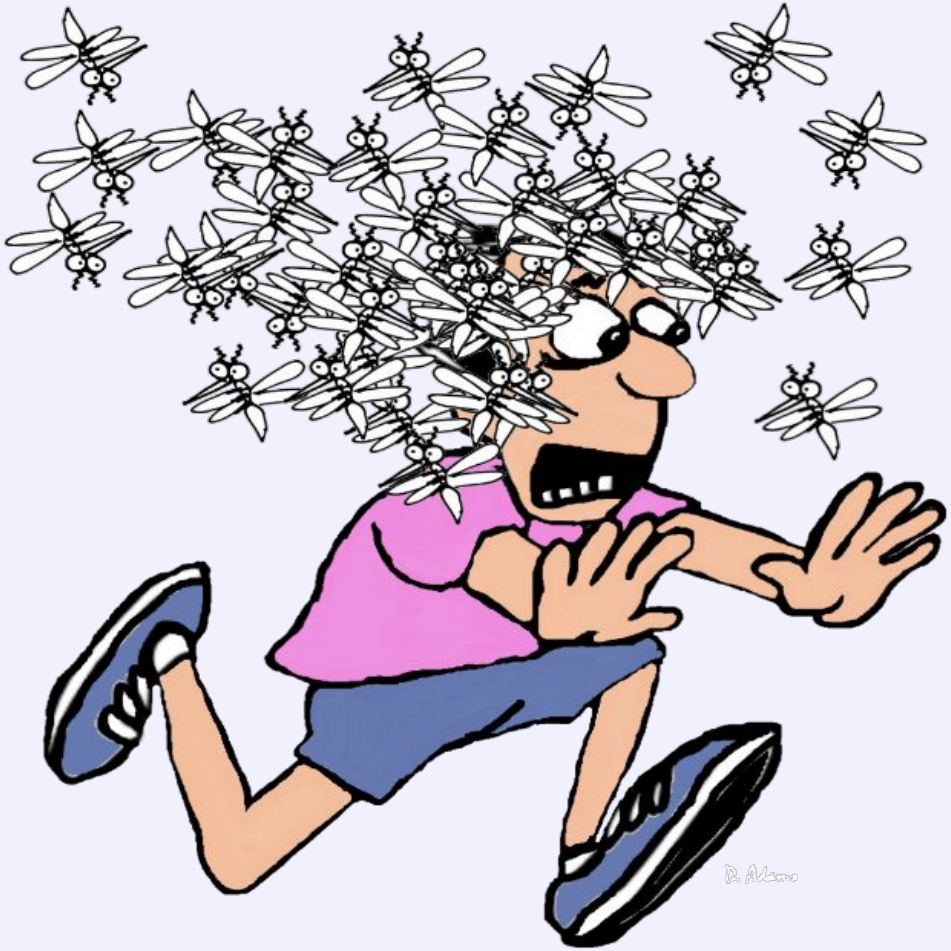
<< prev 1 2 3 4 5 6 7 8 9 10 next >>

Date	D	A	V	Title	Platform	Author
2017-11-13		-		Kirby CMS < 2.5.7 - Cross-Site Scripting	PHP	Ishaq Mohammed
2017-11-13		-		Web Viewer 1.0.0.193 (Samsung SRN-1670D) - Unrestricted File Upload	PHP	0xFFFFFF
2017-11-07		-		ManageEngine Applications Manager 13 - SQL Injection	Windows	Cody Sixteen
2017-11-07		-		pfSense 2.3.1_1 - Command Execution	PHP	s4squatch
2017-11-04		-		WordPress Plugin Userpro < 4.9.17.1 - Authentication Bypass	PHP	Colette...
2017-11-03		-		Logitech Media Server 7.9.0 - 'Radio URL' Cross-Site Scripting	Multiple	Dewank Pant
2017-11-03		-		Logitech Media Server 7.9.0 - 'favorites' Cross-Site Scripting	Multiple	Dewank Pant

Advantage Of Secure Coding

OWASP
The Open Web Application Security Project

- For Developer / Programmer



Are You Familiar with Go?

OWASP
The Open Web Application Security Project

```
flag.Usage = func() {
    fmt.Fprintf(os.Stderr, "gotags version %s\n\n", VERSION)
    fmt.Fprintf(os.Stderr, "Usage: %s [options] file(s)\n\n",
Args[0])
    flag.PrintDefaults()
}


c main() {
flag.Parse()

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

if flag.NArg() == 0 {
```

```
▼ package
    main

▶ imports

▼ constants
    +AUTHOR_EMAIL
    +AUTHOR_NAME
    +NAME
    +URL
    +VERSION

▼ variabl
    -pri
    -prin
    -silent :
    -sortOutput
```

OWASP
The Open Web Application Security Project

	OWASP Top 10 - 2017
→	A1:2017-Injection
→	A2:2017-Broken Authentication
↘	A3:2017-Sensitive Data Exposure
U	A4:2017-XML External Entities (XXE) [NEW]
↘	A5:2017-Broken Access Control [Merged]
↗	A6:2017-Security Misconfiguration
U	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.]

Common Vulnerabilities

- Sql Injection

```
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

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

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

stmt, err := db.Prepare(sql)

err := stmt.Query(id).Scan(&data)
```

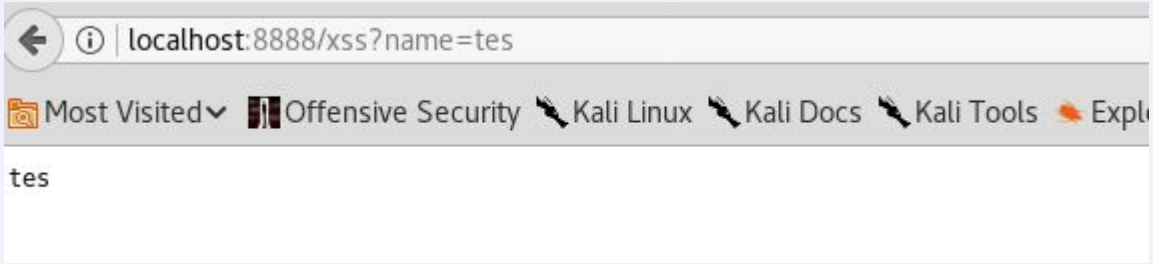
Common Vulnerabilities

- Cross Site Scripting

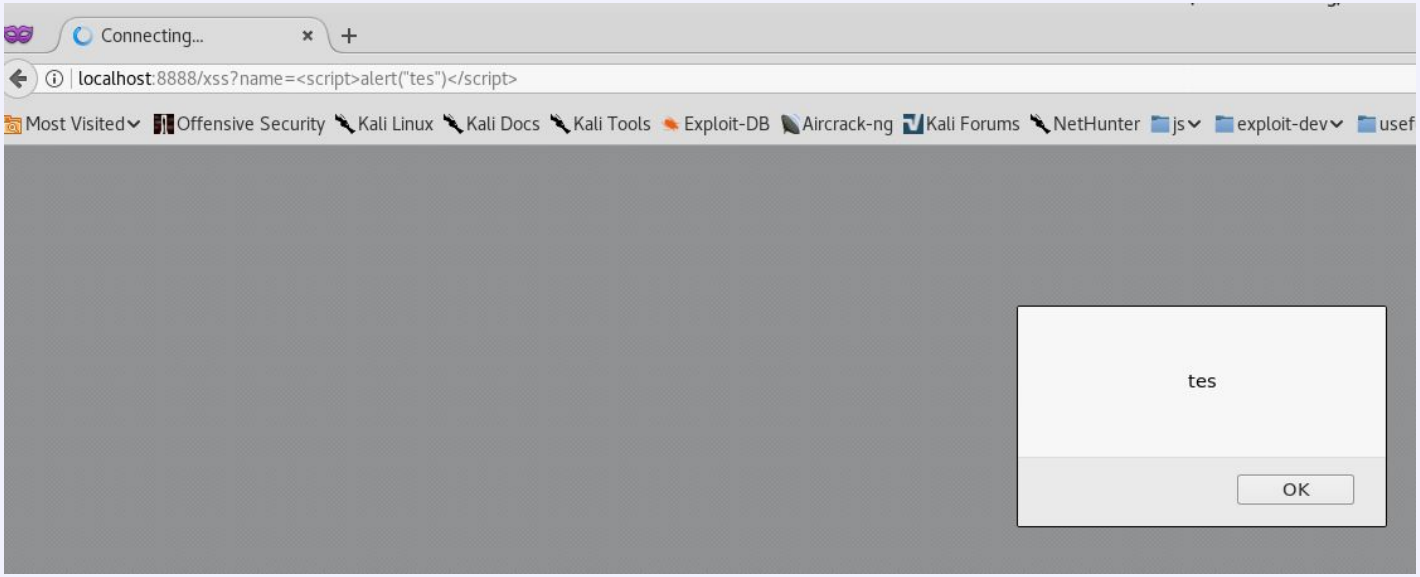
```
func xss(w http.ResponseWriter, r *http.Request){

    name := r.FormValue("name")

     http://localhost:8888/xss?=tes
    io.WriteString(w,name)
}
```

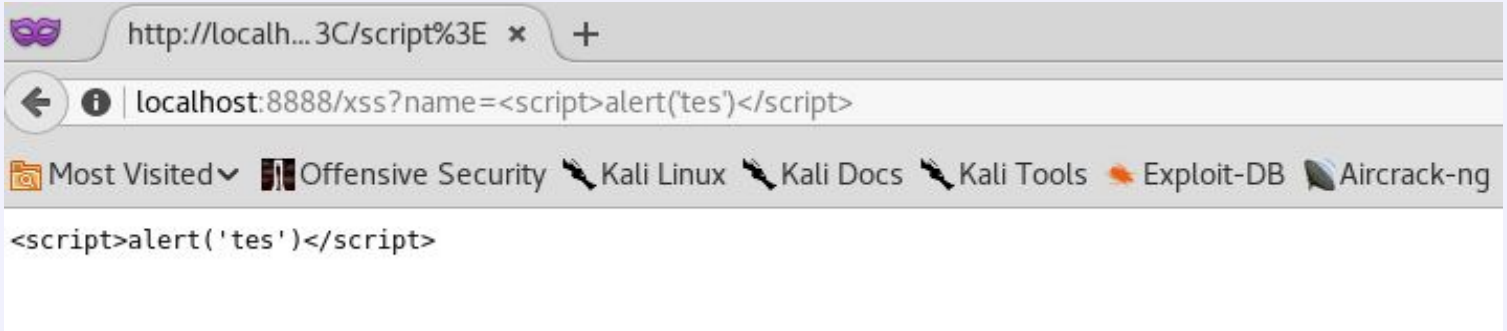


http://localhost:8888/xss?=<script>alert(‘tes’)<script>



- How about implementing content-type?

```
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)
```

OWASP

The Open Web Application Security Project

http://localh... 3C/script%3E

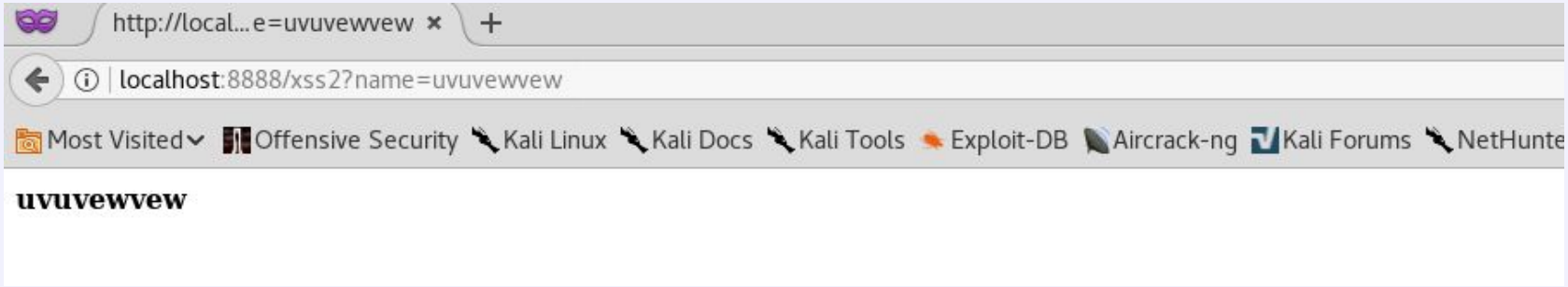
localhost:8888/xss2?name=<script>alert('tes')</script>

Most Visited | Offensive Security | Kali Linux | Kali Docs | Kali Tools | Exploit-DB | Aircrack-ng | Kali

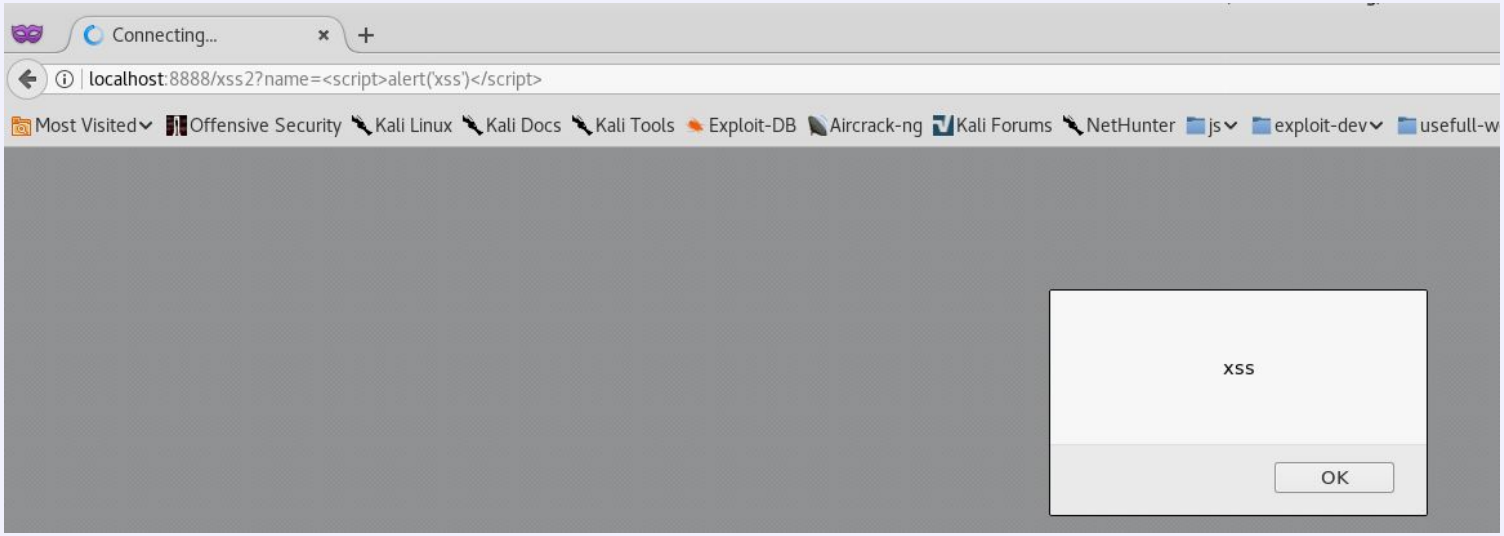
```
&lt;script&gt;alert(&#39;tes&#39;)&lt;/script&gt;
```

- Common Big Mistake

```
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)

```
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

```
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

```
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/verifie
  d")
   }else{
         showErrorAlert()
   }
})
```

OWASP
The Open Web Application Security Project

```
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}
```

OWASP
The Open Web Application Security Project

Handling User Input

When to use input validation, escaping and safeHTML

Require action	Input Validation	Escaping	Safe HTML
Rendered as text	yes	yes	no
Input to be added to javascript	yes	yes	no
Input to be added as a parameter to a URL	yes	yes	no
Rendered as HTML	no	no	yes

Modern XSS Prevention

- Implementing Content Security Policy (CSP)

SQL Injection prevention

- Use of Prepared Statements (with Parameterized Queries).
- White List Input
- Escaping All User Supplied Input
- Enforcing Least Privilege

Authorization (IDOR)

- 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

GoVWA

OWASP
The Open Web Application Security Project

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

OWASP
The Open Web Application Security Project

Thanks.

Any Questions?