

**Please check your attendance
using Blackboard!**

Lecture 7 – Web Security

[COSE451] Software Security

Instructor: Seunghoon Woo

Spring 2024

Overview

- **Cross Site Scripting (XSS)**
- **SQL injection**

Cross Site Scripting (XSS)

- **A vulnerability that can target users of a web page**
 - An attacker inserts malicious scripts into web resources to execute them on the **user's web browser**
 - Mainly occurs in client-side language
 - E.g., JavaScript

2023 CWE Top 25 Most Dangerous Software Weaknesses

[Top 25 Home](#)

Share via: [Twitter](#)

[View in table format](#)

[Key Insights](#)

[Methodology](#)

1

Out-of-bounds Write

[CWE-787](#) | CVEs in KEV: 70 | Rank Last Year: 1

2

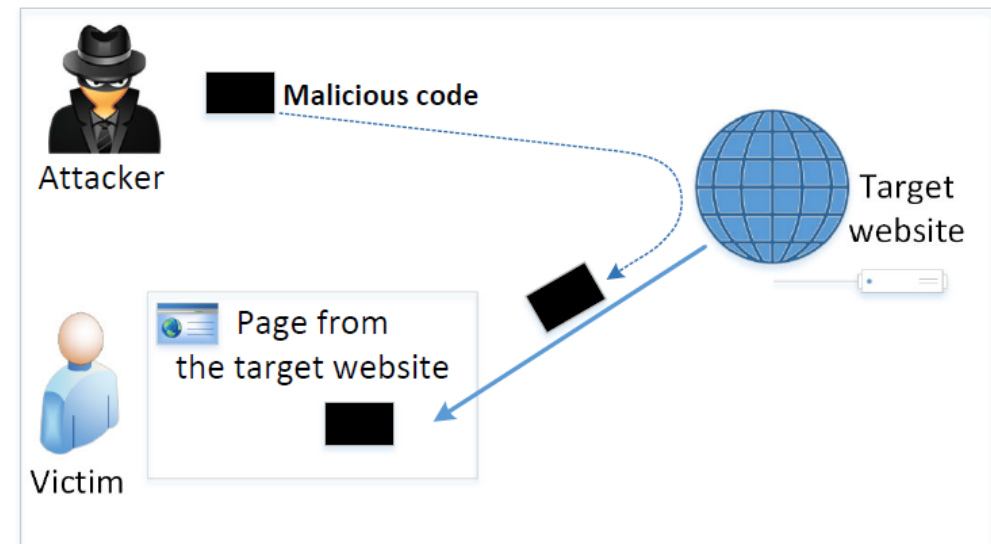
Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

[CWE-79](#) | CVEs in KEV: 4 | Rank Last Year: 2

Cross Site Scripting (XSS)

- **A vulnerability that can target users of a web page**

1. Assume that XSS vulnerability exists in a normal web page
2. Attackers can create malicious posts using XSS attacks
3. When a user visits a page containing a malicious script, data can be leaked
 - Cookies, sessions, etc.



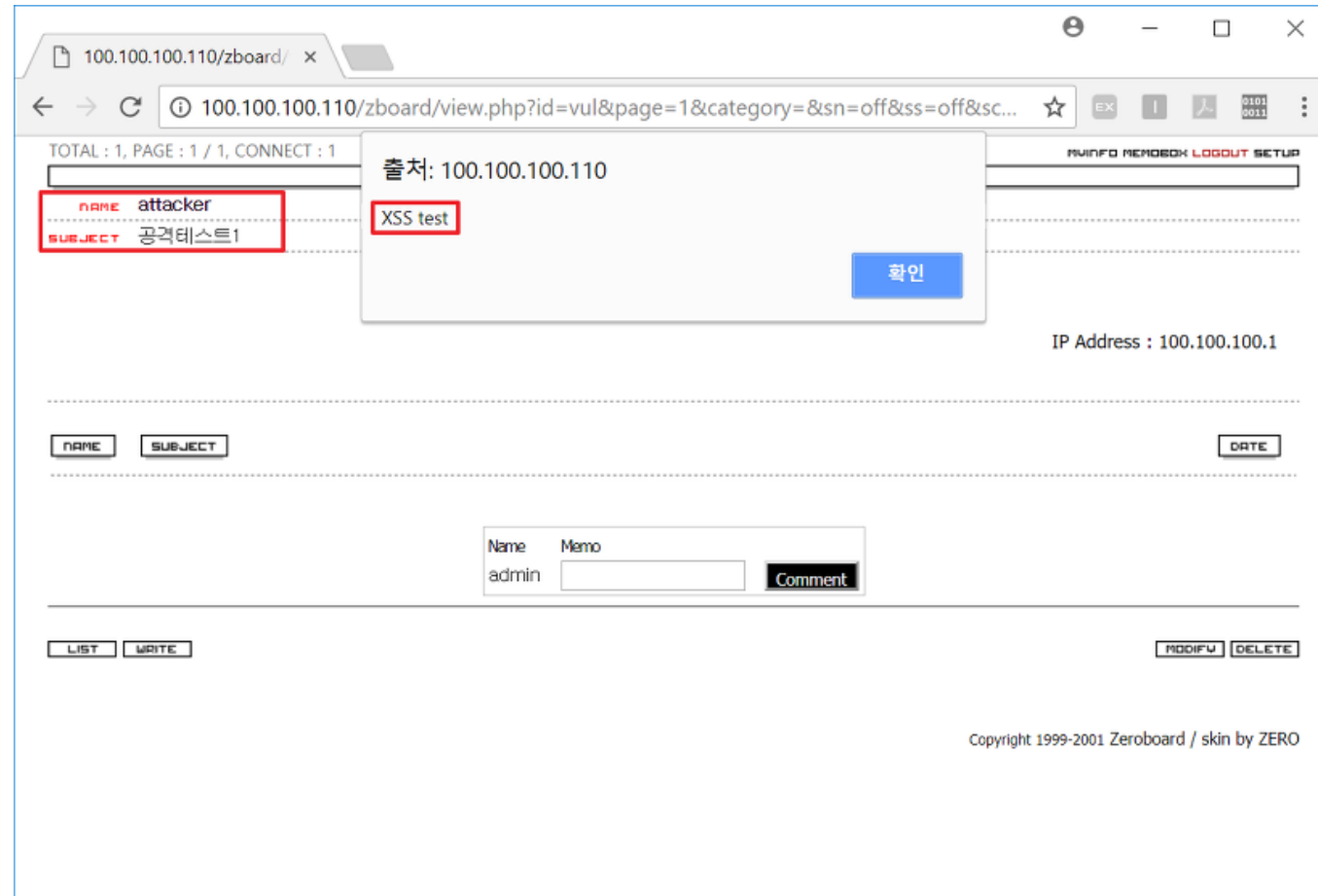
Cross Site Scripting (XSS)

- Example



Cross Site Scripting (XSS)

- Example



Cross Site Scripting (XSS)

- **Types of XSS**

- Reflected XSS attack
- Stored XSS attack

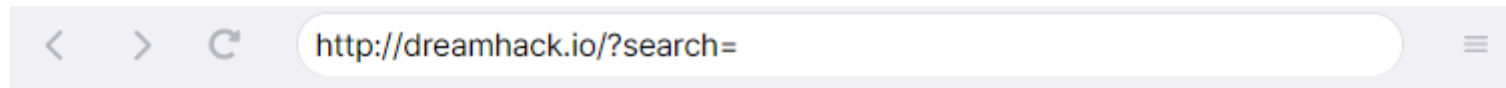
Cross Site Scripting (XSS)

- **Reflected XSS attack**

- Occurs when the server **outputs** a request containing a malicious script
- The user's input value passed through the HTTP request is included in the response and sent back to the user
 - Thus it is called "reflected"

Cross Site Scripting (XSS)

- **Reflected XSS attack**



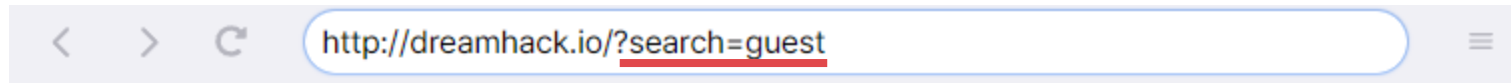
All results

Id	Nickname
guest	Guest
admin	Administrator

```
1 <h3>
2   All results
3 </h3>
4 <table>
5   <tr>
6     ...
7   </tr>
8 </table>
```

Cross Site Scripting (XSS)

- Reflected XSS attack



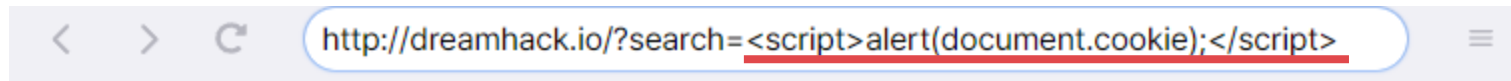
Search result for guest

Id	Nickname
guest	Guest

```
1 <h3>
2   Search result for guest
3 </h3>
4 <table>
5   <tr>
6     ...
7   </tr>
8 </table>
```

Cross Site Scripting (XSS)

- **Reflected XSS attack**



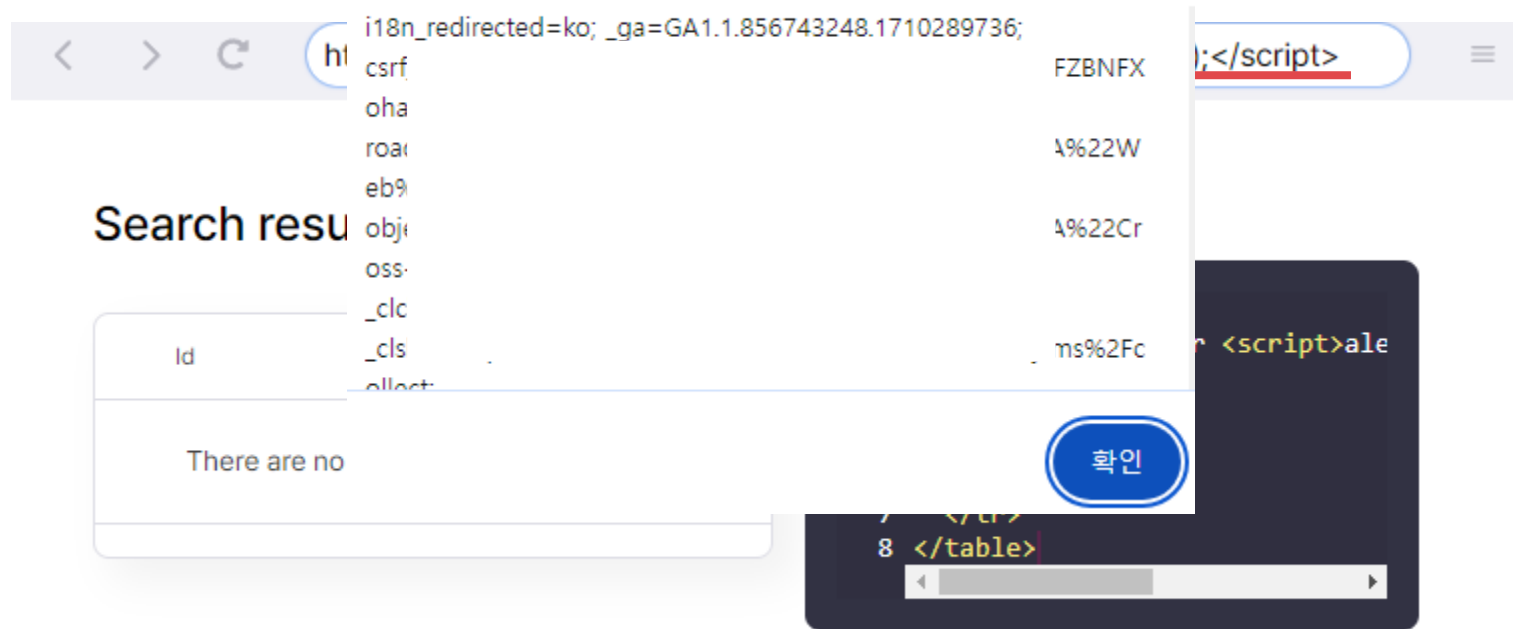
Search result for

Id	Nickname
There are no records matching your request	

```
1 <h3>
2   Search result for <script>ale
3 </h3>
4 <table>
5   <tr>
6     ...
7   </tr>
8 </table>
```

Cross Site Scripting (XSS)

- Reflected XSS attack



Cross Site Scripting (XSS)

- **Reflected XSS attack**

- Attack scenario

- 1) A malicious user has discovered a site with weak security (XSS exists)
- 2) Creates a URL containing a script that can steal user information from a site with weak security and delivers it to general users as spam email
- 3) General users click on the URL link sent via email
- 4) The malicious script is executed while sending the response message to the user's browser
- 5) User information is passed to malicious users through malicious scripts

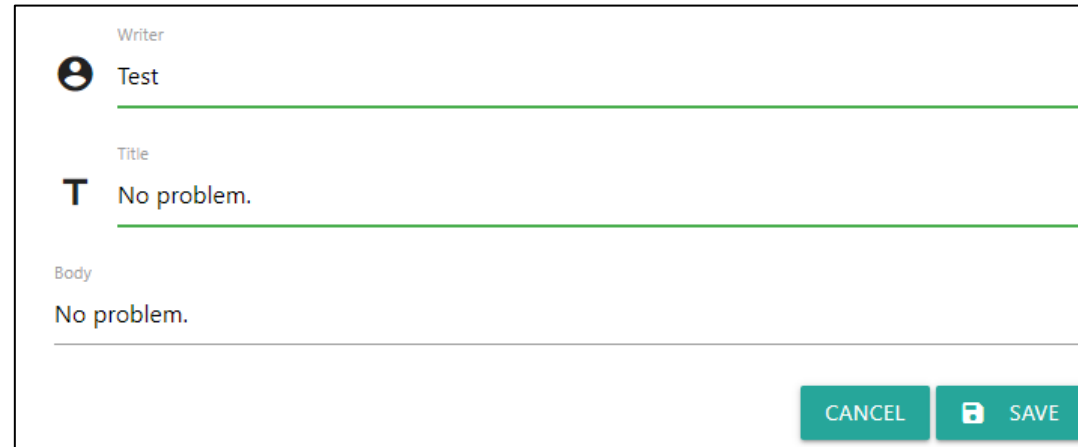
Cross Site Scripting (XSS)

- **Stored XSS attack**

- Occurs when searching for malicious scripts **stored** in the server database or file
 - Uploading malicious posts, comments with malicious scripts

Cross Site Scripting (XSS)

- **Stored XSS attack**




The screenshot shows a web form with three input fields: "Writer", "Title", and "Body". The "Writer" field contains the text "Test". The "Title" field contains the text "No problem." with a large "T" icon to its left. The "Body" field also contains the text "No problem.". At the bottom right of the form, there are two buttons: "CANCEL" and "SAVE".

Cross Site Scripting (XSS)

- **Stored XSS attack**


No	Title	Writer
1	No problem.	Test



 Test

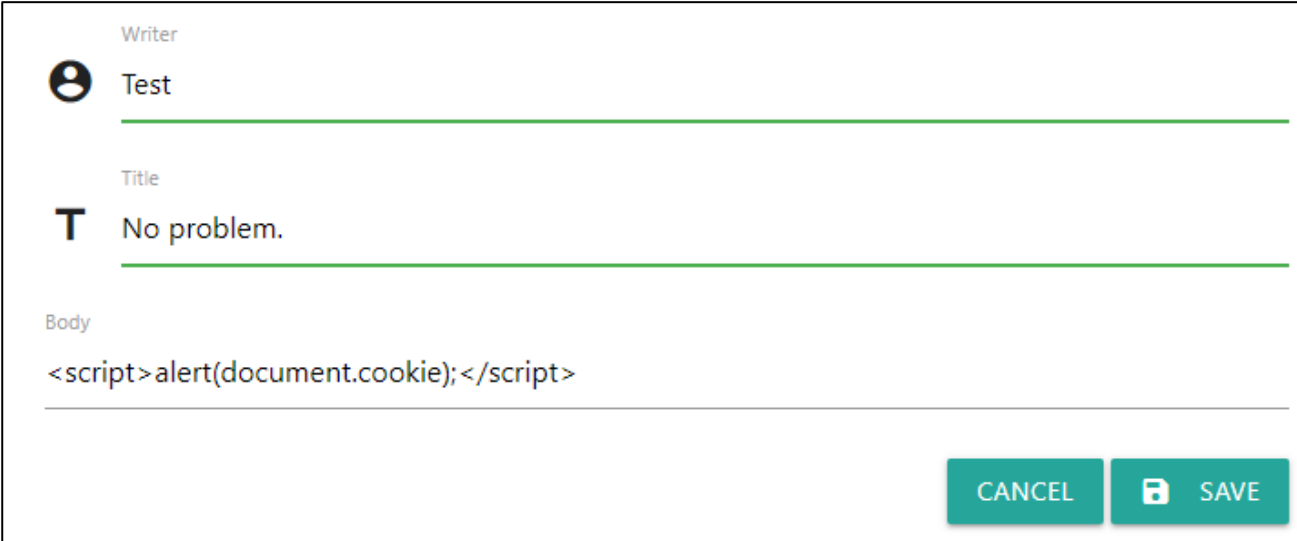
T No problem.

No problem.



Cross Site Scripting (XSS)

- **Stored XSS attack**



Writer
Test

Title
No problem.

Body
<script>alert(document.cookie);</script>

CANCEL SAVE

Cross Site Scripting (XSS)

- **Stored XSS attack**

learn.dreamhack.io 내용:

```
i18n_redirected=ko; _ga=GA1.1.856743248.1710289736;
csrf FZBNFX
oha
roac A%22W
eb%
obje A%22Cr
oss-
_clc
_clc ms%2Fc
ollect
```

Test

No problem.

확인

LIST

Cross Site Scripting (XSS)

- **Stored XSS attack**

- Attack scenario

- 1) A malicious user has discovered a site with weak security (XSS exists)
- 2) Write and post a script that can steal user information on a bulletin board provided by a site with weak security
- 3) When an ordinary user reads a post written by a malicious user, (s)he receives a post response containing a malicious script from the server
- 4) The malicious script is executed while sending the response message to the user's browser
- 5) User information is passed to malicious users through malicious scripts

Cross Site Scripting (XSS)

- **Prevent XSS**

- **Input Validation**

- Validate user input in the web application
 - To ensure that it does not contain unsafe characters or scripts
 - This helps prevent malicious scripts from being passed to the server

- **Escape Input**

- Before outputting user input, perform escaping to encode special characters
 - To prevent scripts from being executed
 - This ensures that any injected scripts are treated as data, not executable code

Cross Site Scripting (XSS)

- Prevent XSS
 - Escape Input

```
function escapeHtml(unsafe) {  
    return unsafe  
        .replace(/&/g, "&amp;")  
        .replace(/</g, "&lt;")  
        .replace(/>/g, "&gt;")  
        .replace(/"/g, "&quot;")  
        .replace(/'/g, "&#039;");  
}
```

SQL injection

- **SQL injection**

Command injection

- **Injection**

- Injecting malicious data into a program to execute it as system commands, code, database queries, etc.

- **Command injection**

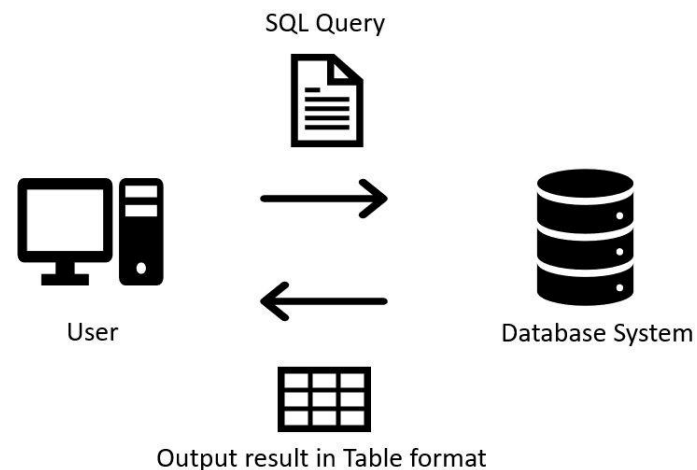
- Executing user input as system commands
- Arbitrary commands may be executed if user input is not properly validated

SQL injection

- **SQL injection**

- SQL?

- Structured Query Language
 - A data processing language used to extract and manipulate data from databases



SQL injection

- **SQL injection**

- Example

userID	name	score
SHW	Seunghoon Woo	100
KSY	Kyeongseok Yang	98
...		

midtermScore Table

SQL injection

- **SQL injection**

- Example

userID	name	score
SHW	Seunghoon Woo	100
KSY	Kyeongseok Yang	98
...		

midtermScore Table

- **SELECT**
 - To specify the columns that we want to retrieve data from
- **FROM**
 - To specify the table from which we want to retrieve the data
- **WHERE**
 - To filter the rows retrieved from the table based on conditions

SQL injection

- **SQL injection**

- Example

userID	name	score
SHW	Seunghoon Woo	100
KSY	Kyeongseok Yang	98
...		

midtermScore Table

```
SELECT      score
FROM        midtermScore
WHERE       name = 'Seunghoon Woo'
```

```
SELECT score FROM midtermScore WHERE name='Seunghoon Woo'
```

SQL injection

- **SQL injection**

- Example

userID	name	score
SHW	Seunghoon Woo	100
KSY	Kyeongseok Yang	98
...		

midtermScore Table

```
SELECT      score
FROM        midtermScore
WHERE       name = 'Seunghoon Woo'
```

```
SELECT score FROM midtermScore WHERE name='Seunghoon Woo'
> 100
```

SQL injection

- SQL injection

- Example

- Command injection

```
seunghoonwoo@seunghoonwoo-virtual-machine:~$ ./ping_server
Enter IP: 127.0.0.1 ; cat /etc/passwd
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.033 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.038 ms

--- 127.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.033/0.035/0.038/0.002 ms
root:x:0:0:root:
daemon:x:1:1:da
bin:x:2:2:bin:/t
```

SQL injection

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table

SQL injection

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table

LOGIN
Please enter your login and password!

Username

Password

[Forgot password?](#)

Login



`SELECT userID FROM userInfo WHERE userID = " and userPW = "`

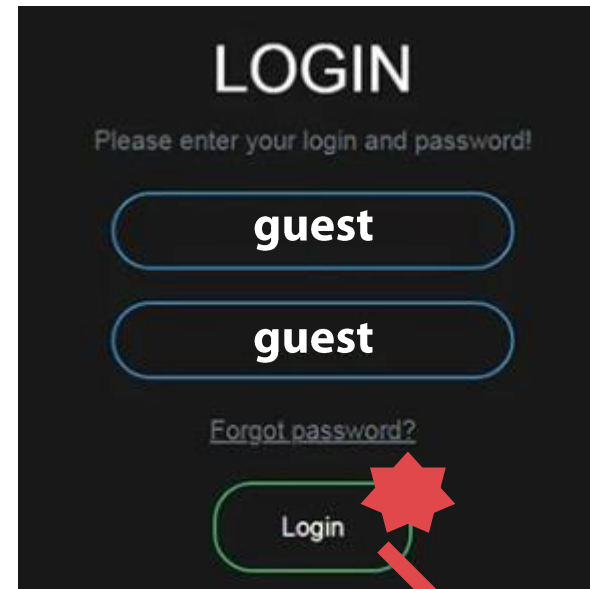
SQL injection

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



`SELECT userID FROM userInfo WHERE userID = 'guest' and userPW = 'guest'`

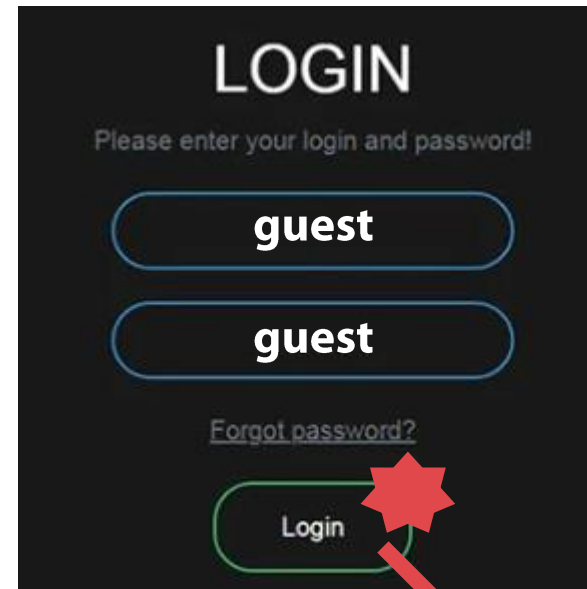
SQL injection

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



`SELECT userID FROM userInfo WHERE userID = 'guest' and userPW = 'guest' > guest`

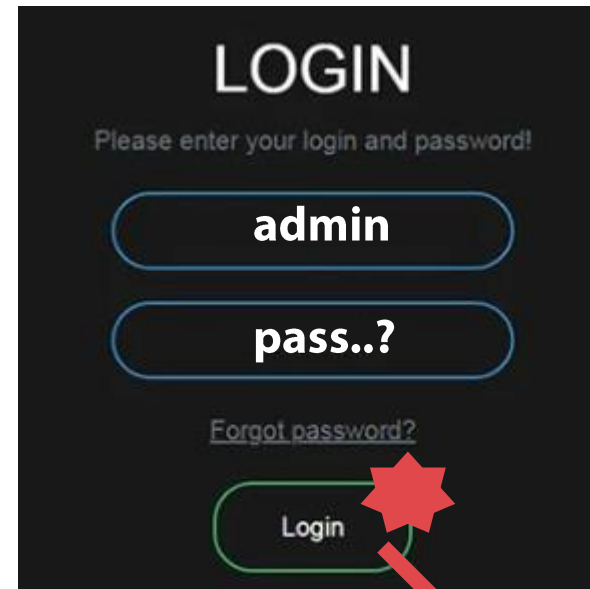
SQL injection

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



`SELECT userID FROM userInfo WHERE userID = 'admin' and userPW = 'pass..?'`

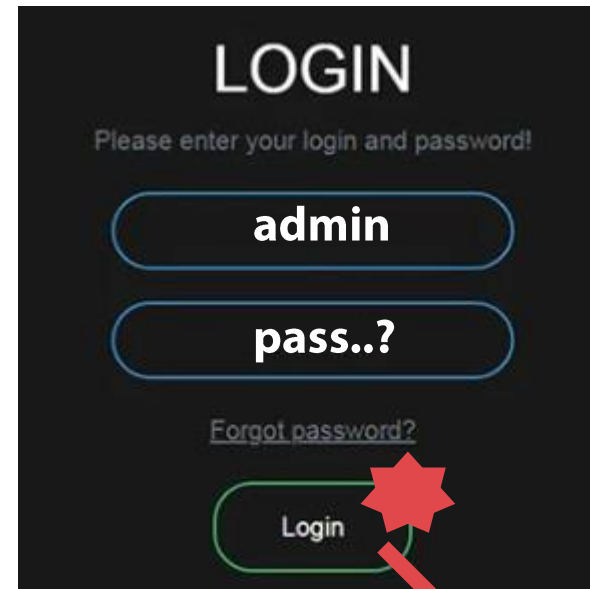
SQL injection

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



`SELECT userID FROM userInfo WHERE userID = 'admin' and userPW = 'pass..?'`

`>`

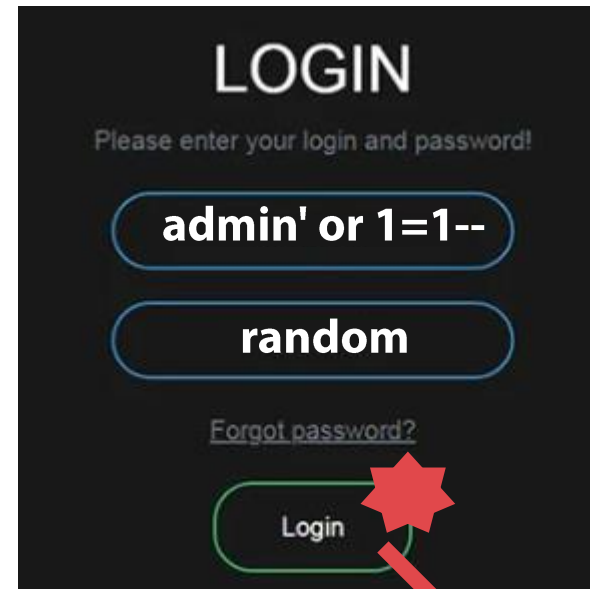
SQL injection

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



`SELECT userID FROM userInfo WHERE userID = 'admin' or 1=1 --' and userPW = 'random'`

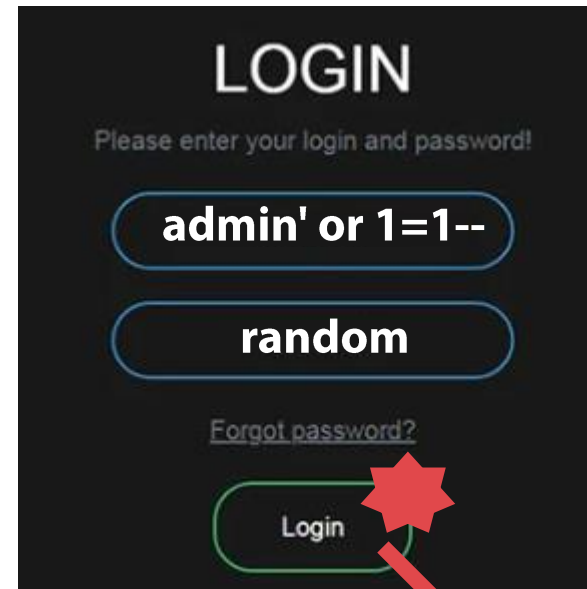
SQL injection

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



`SELECT userID FROM userInfo WHERE userID = 'admin' or 1=1 --' and userPW = 'random'`
`> guest, admin`

SQL injection

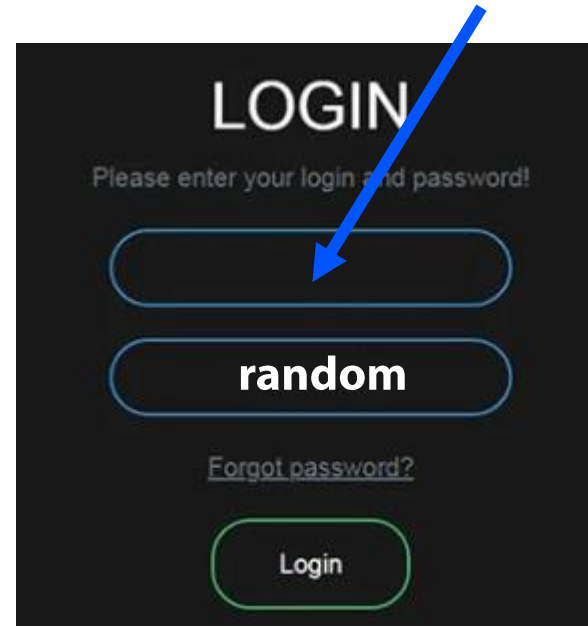
admin' or 1=1; **SELECT** userPW **FROM** userInfo --

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



SQL injection

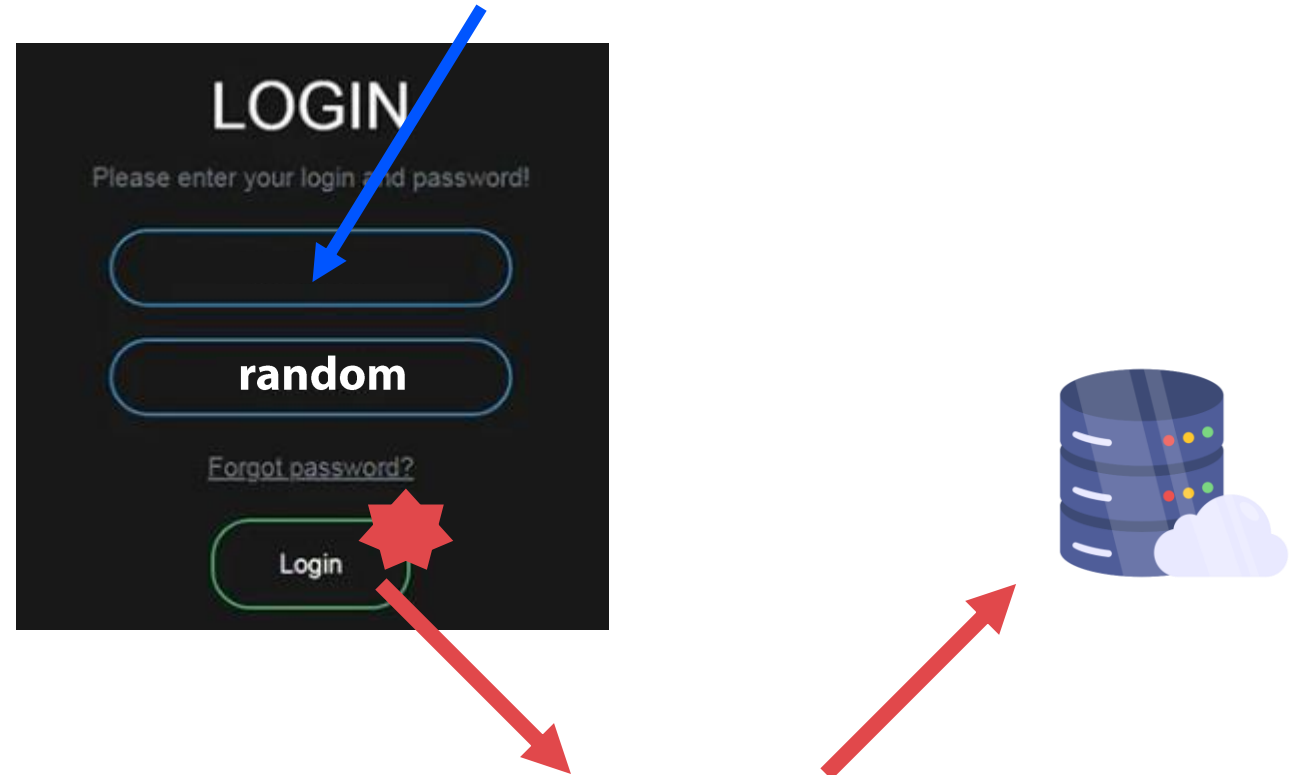
admin' or 1=1; **SELECT** userPW **FROM** userInfo --

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



SELECT userID **FROM** userInfo **WHERE** userID = 'admin' or 1=1;
SELECT userPW **FROM** userInfo --' and userPW = 'random'

SQL injection

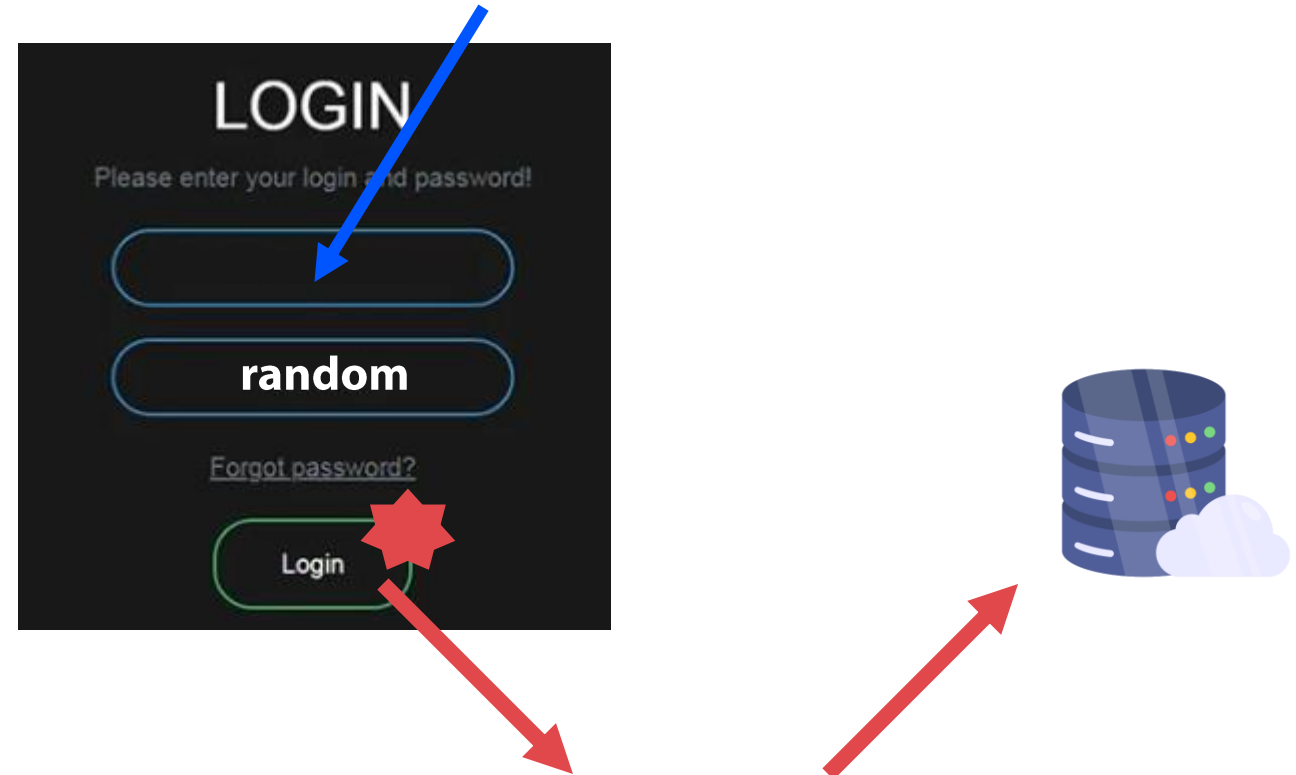
admin' or 1=1; **SELECT** userPW **FROM** userInfo --

- **SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



```
SELECT userID FROM userInfo WHERE userID = 'admin' or 1=1;  
SELECT userPW FROM userInfo --' and userPW = 'random'  
> {"userID": "guest"} {"userPW": "guest"} {"userID": "admin"} {"userPW": "secretsecret"}
```


SQL injection

- **Blind SQL injection**

- Maliciously retrieve data from a database
- Kind of a twenty questions game (스무고개)

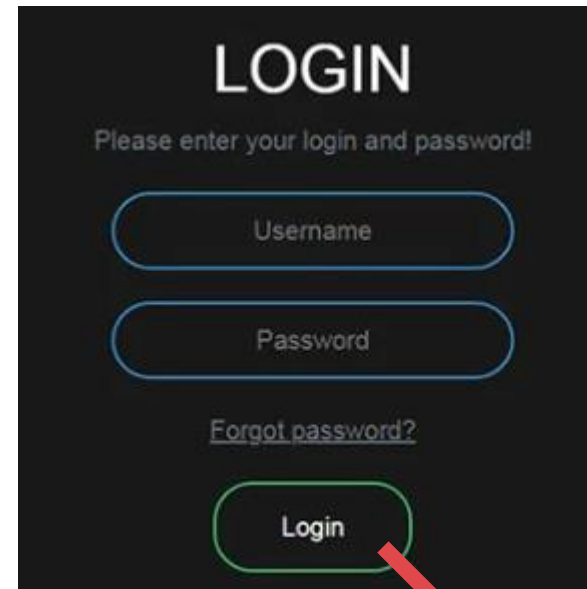
SQL injection

- **Blind SQL injection**

- Example

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



Is the **first** letter of the password for the admin account 'r'? > **FALSE**
Is the **first** letter of the password for the admin account 's'? > **TRUE**
Is the **second** letter of the password for the admin account 'd'? > **FALSE**
Is the **second** letter of the password for the admin account 'e'? > **TRUE**

SQL injection

Binary ↕	Oct ↕	Dec ↕	Hex	Glyph		
				1963 ↕	1965 ↕	1967 ↕
110 0001	141	97	61		a	
110 0010	142	98	62		b	
110 0011	143	99	63		c	
110 0100	144	100	64		d	
110 0101	145	101	65		e	

- **Blind SQL injection**

- Using two functions: `ascii` and `substr`

- `ascii`

- A function that returns the passed character in ASCII format
 - E.g., Running `ascii('a')` returns 97

- `substr`

- Gets the value from the specified position in the string up to its length
 - E.g., `substr(string, position, length)`
 - `substr('ABCD', 1, 1) = 'A'`
 - `substr('ABCD', 2, 2) = 'BC'`

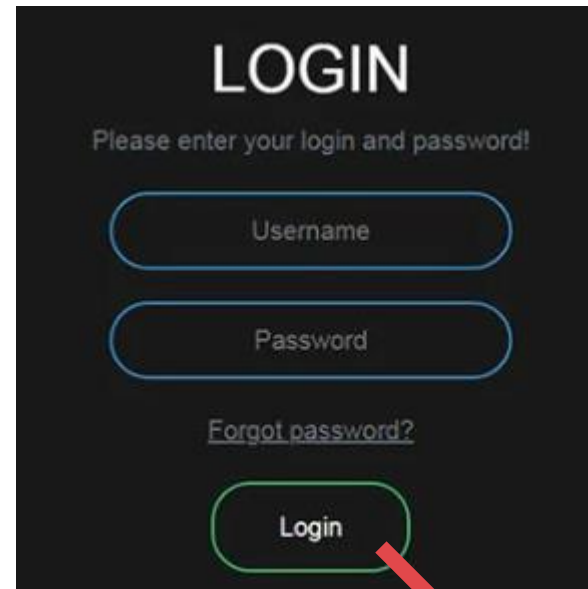
SQL injection

Binary	Oct	Dec	Hex	Glyph		
				1963	1965	1967
111 0010	162	114	72			r
111 0011	163	115	73			s
111 0100	164	116	74			t
111 0101	165	117	75			u

- **Blind SQL injection**

userID	userPW
guest	guest
admin	*****
...	

userInfo Table



`SELECT userID FROM userInfo WHERE userID = 'admin' and ascii(substr(userPW,1,1))=114 --' and... >`
`SELECT userID FROM userInfo WHERE userID = 'admin' and ascii(substr(userPW,1,1))=115 --' and... > admin`

SQL injection

- **Prevent SQL injection**

- Input validation

- String filtering: filters and escapes special strings used in SQL injection from user input
 - Make SQL queries safe by escaping characters such as single quotes (') and semicolons (;)
 - Type check of input value: when receiving user input, check whether it is an appropriate type
 - E.g., a numeric field accepts only numeric values, and a string field accepts string values

- Using parameterized queries

- `SELECT * FROM table WHERE condition=?` → `SELECT * FROM table WHERE condition=value`

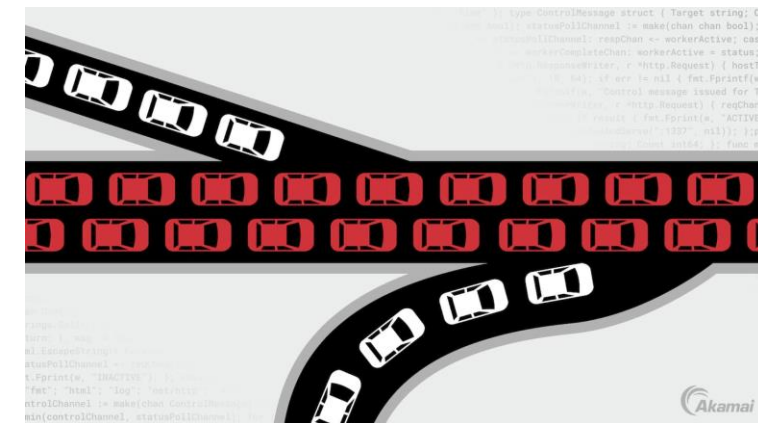
Attack vectors

- **We should separate the means of attack & the goals of the attack**
- **Attacker goals can be grouped into three broad classes**
 1. Denial of service (DoS)
 2. Leaking information
 3. Privilege escalation

Attack vectors

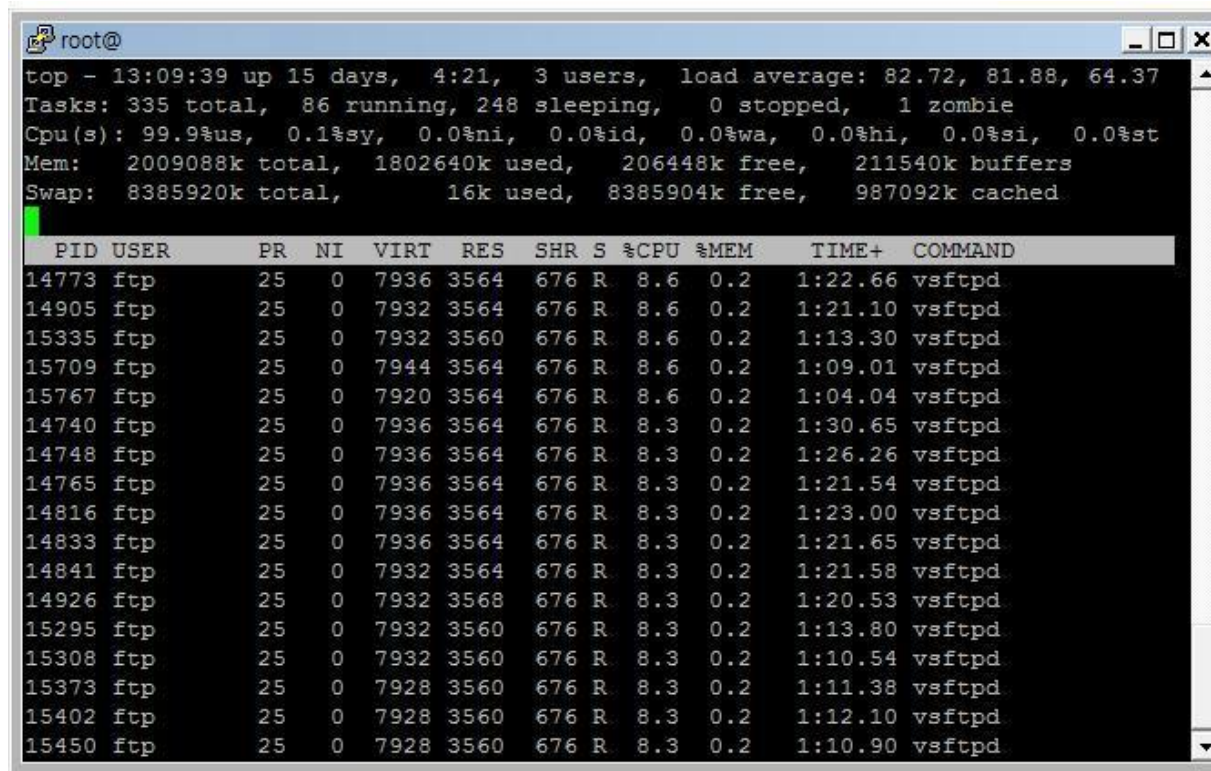
- **Denial of service (DoS)**

- To disrupt the **availability** of critical services
- The main security concern in network environments
- In the software security
 - It can be made to consume all disk space, memory, and CPU usage



Attack vectors

- Denial of service (DoS): CVE-2011-0762



```
root@
top - 13:09:39 up 15 days, 4:21, 3 users, load average: 82.72, 81.88, 64.37
Tasks: 335 total, 86 running, 248 sleeping, 0 stopped, 1 zombie
Cpu(s): 99.9%us, 0.1%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2009088k total, 1802640k used, 206448k free, 211540k buffers
Swap: 8385920k total, 16k used, 8385904k free, 987092k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 14773 ftp        25   0  7936 3564  676  R   8.6   0.2   1:22.66 vsftpd
 14905 ftp        25   0  7932 3564  676  R   8.6   0.2   1:21.10 vsftpd
 15335 ftp        25   0  7932 3560  676  R   8.6   0.2   1:13.30 vsftpd
 15709 ftp        25   0  7944 3564  676  R   8.6   0.2   1:09.01 vsftpd
 15767 ftp        25   0  7920 3564  676  R   8.6   0.2   1:04.04 vsftpd
 14740 ftp        25   0  7936 3564  676  R   8.3   0.2   1:30.65 vsftpd
 14748 ftp        25   0  7936 3564  676  R   8.3   0.2   1:26.26 vsftpd
 14765 ftp        25   0  7936 3564  676  R   8.3   0.2   1:21.54 vsftpd
 14816 ftp        25   0  7936 3564  676  R   8.3   0.2   1:23.00 vsftpd
 14833 ftp        25   0  7936 3564  676  R   8.3   0.2   1:21.65 vsftpd
 14841 ftp        25   0  7932 3564  676  R   8.3   0.2   1:21.58 vsftpd
 14926 ftp        25   0  7932 3568  676  R   8.3   0.2   1:20.53 vsftpd
 15295 ftp        25   0  7932 3560  676  R   8.3   0.2   1:13.80 vsftpd
 15308 ftp        25   0  7932 3560  676  R   8.3   0.2   1:10.54 vsftpd
 15373 ftp        25   0  7928 3560  676  R   8.3   0.2   1:11.38 vsftpd
 15402 ftp        25   0  7928 3560  676  R   8.3   0.2   1:12.10 vsftpd
 15450 ftp        25   0  7928 3560  676  R   8.3   0.2   1:10.90 vsftpd
```

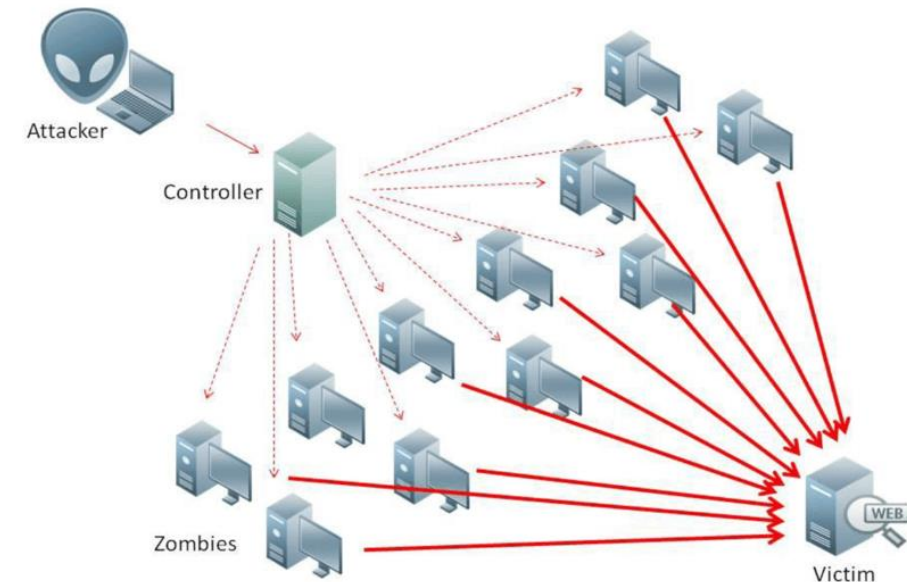

Attack vectors

- **Distributed Denial of service (DDoS)**

- A malicious attempt to disrupt normal traffic of a targeted server, service, or network by overwhelming it with a flood of internet traffic

- **Distributed?**

- DDoS attacks involve multiple sources (bots)
 - Compromised computers, IoT devices, etc.



Attack vectors

- **Information Leakage**

- To disrupt the **confidentiality** of critical services
- Abnormal or unintended transfers of sensitive information to the attacker
 - E.g., Personal information, money, software code, government secrets
- Consequences
 - Reputation damage: loss of trust from customers and clients
 - Financial losses
 - Operational disruption

Attack vectors

- **Information Leakage**

- Example: Yahoo data breach

- In 2013 and 2014 Internet service company Yahoo was subjected to data breaches
 - More than 3 billion account information leaked
 - Fortunately, the attacker accessed account information such as security questions and answers, but was unable to steal general text passwords, payment cards, and bank data



Attack vectors

- **Privilege escalation**
 - Please refer to the lecture notes 5 and 6!



Next Lecture

- **Class review (for midterm exam!)**