CS348
Web Security

Min Suk Kang & Kihong Heo

KAIST

Web Security

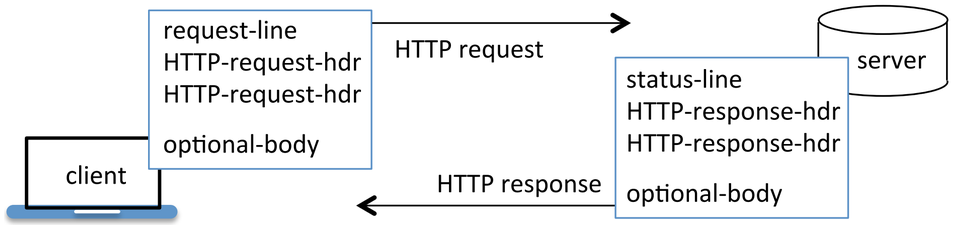
- What can go wrong on the web through browser-server interactions?
  - A lot of sensitive data through the web (e.g., banking)
  - A lot of non-trivial code through the web (e.g., JavaScript, SQL Queries)
- Broad impacts on many users worldwide

URL

- Uniform Resource Locator
- Global identifiers of network-retrievable documents
- Example

HTTP

- Hypertext Transfer Protocol
- The primary protocol for data transfer between web browsers and servers
- A stateless protocol: no state is retained across successive requests



<request-method> <request-URI> <HTTP-version>

request-line (example): `GET /filepath/file.html HTTP/1.1`

HTTP-request-hdr (examples):

```
Referer: http://from.domain.com/doc.html
Host: target.domain.com
User-Agent: Mozilla/5.0
```

<keyword>: <value>

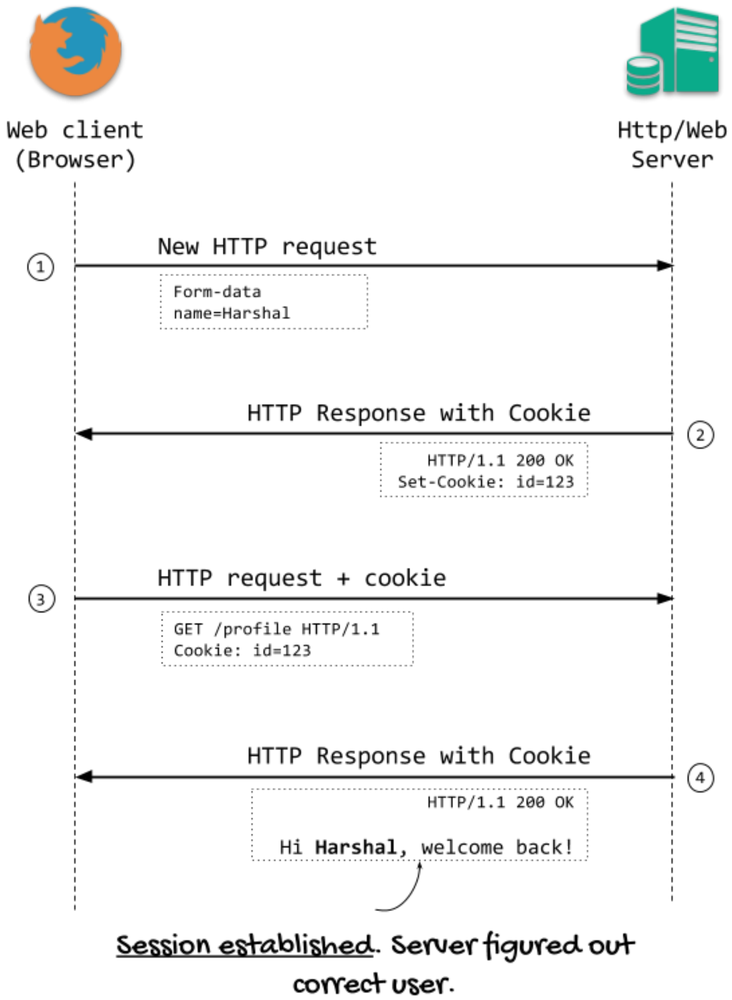
status-line (examples):

```
HTTP/1.1 200 OK
HTTP/1.1 404 Not Found
```

*Figure from Oorschot's book

Cookies

- What happens if all web services are stateless?
- Idea: store a server-created file (cookie) in the browser
- Examples
  - Authentication (log in)
  - Personalization (language preference, shopping cart)
  - User tracking



HTML

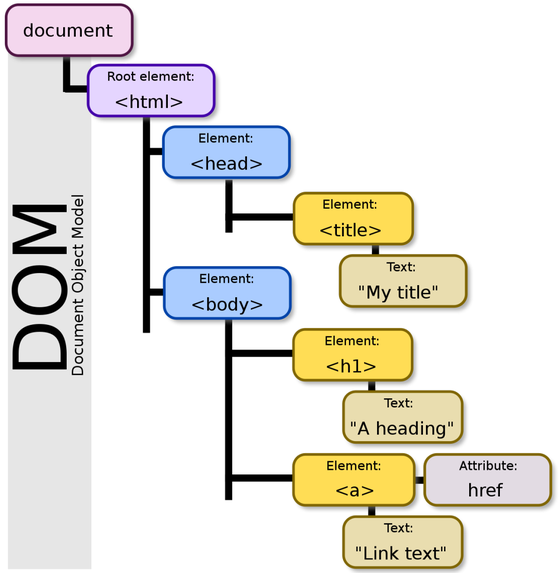
- Hypertext Markup Language (i.e., for layout)
  - NOT programming language (i.e., for computation)!
- A web page (document) is written in HTML using markup tags
  - E.g., <p>, <img>
- Describes a hyper-text document
  - E.g., image, audio, video
- What if we need computation?

Script

- HTML may contain program code written in a scripting language
  - Make web pages more dynamic
  - Most popular: JavaScript
- Web browsers execute the script program
  - A webpage gets to execute potentially malicious code on your machine!

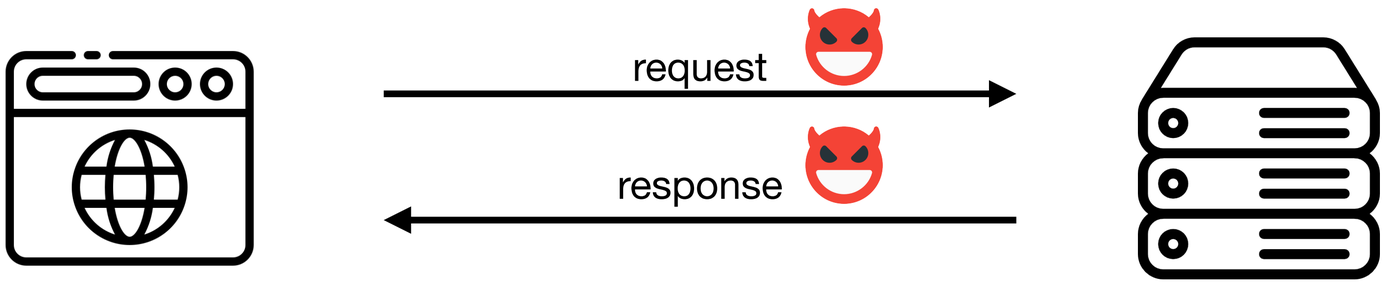
DOM

- Document Object Model
- An HTML document: structured data
- Property: content in DOM tree
  - Can be modified by JavaScript



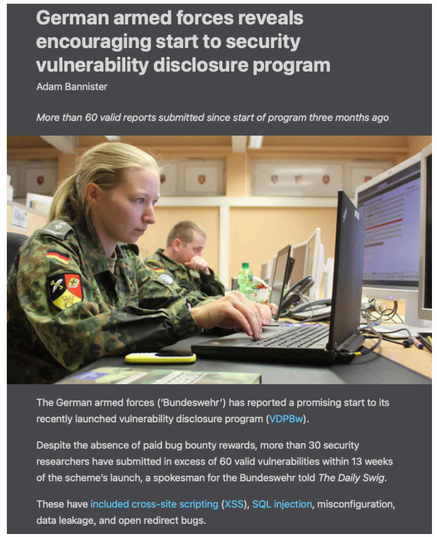
SW on the Web

- Mix of server-side and client-side code
  - Server-side code (e.g., PHP, Ruby, etc) runs on the web server
  - Client-side code (e.g., JavaScript, etc) runs in the web browser



Common Web Security Attacks

- Common examples
  - SQL injection
  - Cross-site script (XSS)
  - Cross-stie Request Forgery (CSRF)



German armed forces reveals encouraging start to security vulnerability disclosure program

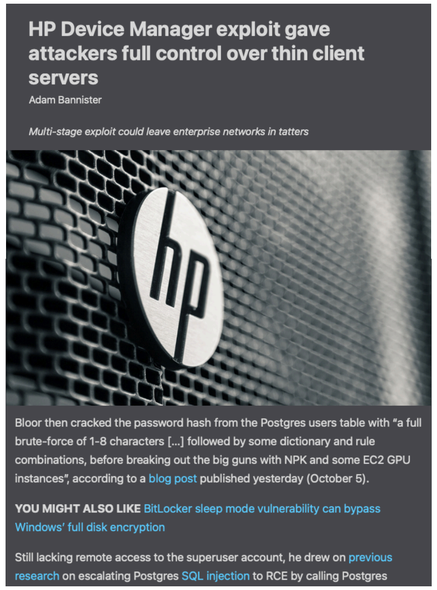
Adam Bannister

More than 60 valid reports submitted since start of program three months ago

The German armed forces ('Bundeswehr') has reported a promising start to its recently launched vulnerability disclosure program (VDPBw).

Despite the absence of paid bug bounty rewards, more than 30 security researchers have submitted in excess of 60 valid vulnerabilities within 13 weeks of the scheme's launch, a spokesman for the Bundeswehr told *The Daily Swig*.

These have included cross-site scripting (XSS), SQL injection, misconfiguration, data leakage, and open redirect bugs.



HP Device Manager exploit gave attackers full control over thin client servers

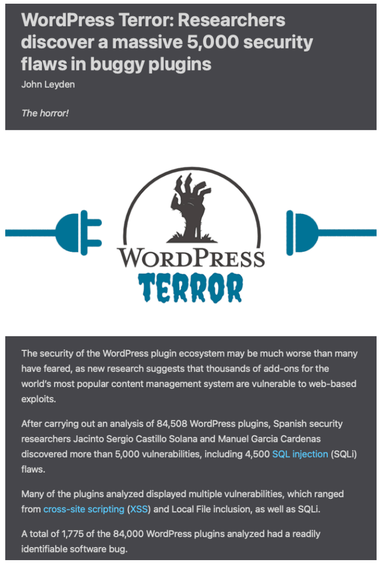
Adam Bannister

Multi-stage exploit could leave enterprise networks in tatters

Bloor then cracked the password hash from the Postgres users table with "a full brute-force of 1-8 characters [...] followed by some dictionary and rule combinations, before breaking out the big guns with NPK and some EC2 GPU instances", according to a blog post published yesterday (October 5).

YOU MIGHT ALSO LIKE BitLocker sleep mode vulnerability can bypass Windows' full disk encryption

Still lacking remote access to the superuser account, he drew on previous research on escalating Postgres SQL injection to RCE by calling Postgres



WordPress Terror: Researchers discover a massive 5,000 security flaws in buggy plugins

John Leyden

The horror!

WordPress TERROR

The security of the WordPress plugin ecosystem may be much worse than many have feared, as new research suggests that thousands of add-ons for the world's most popular content management system are vulnerable to web-based exploits.

After carrying out an analysis of 84,508 WordPress plugins, Spanish security researchers Jacinto Sergio Castillo Solana and Manuel Garcia Cardenas discovered more than 5,000 vulnerabilities, including 4,500 SQL injection (SQLi) flaws.

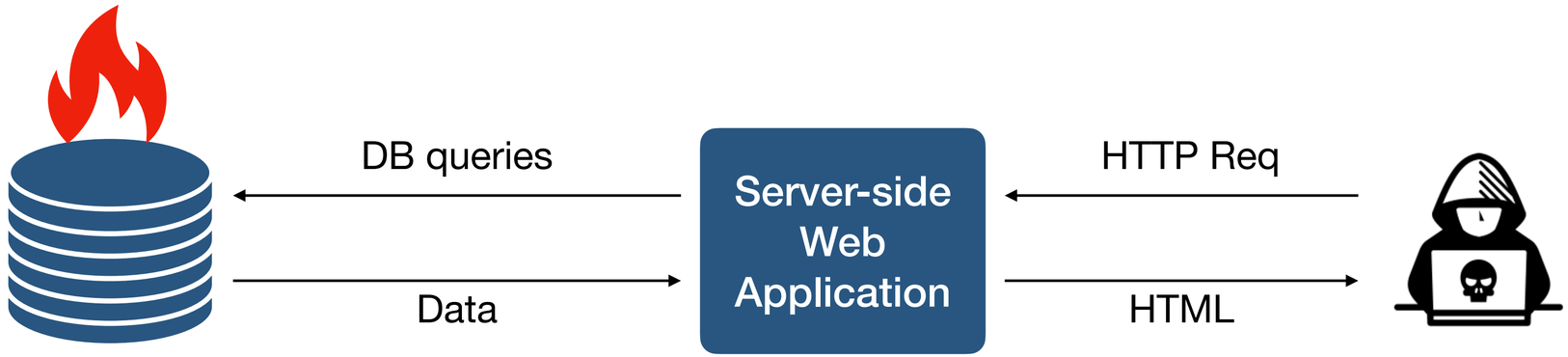
Many of the plugins analyzed displayed multiple vulnerabilities, which ranged from cross-site scripting (XSS) and Local File inclusion, as well as SQLi.

A total of 1,775 of the 84,000 WordPress plugins analyzed had a readily identifiable software bug.

SQL Injection

SQL Injection

- Common security issues for most web applications that store data in DB
  - SQL queries are dynamically constructed by input
- Attack: maliciously manipulate DB via attacker-chosen SQL queries



Example

- A user can log into a web site with a username `name` and password `pw`
- An SQL query string is constructed as follows:

```
query = "SELECT * FROM pswdtab WHERE username='" + name + "' AND password='" + pw + "'"
```

retrieve all fields

from this table

if each row satisfies this condition

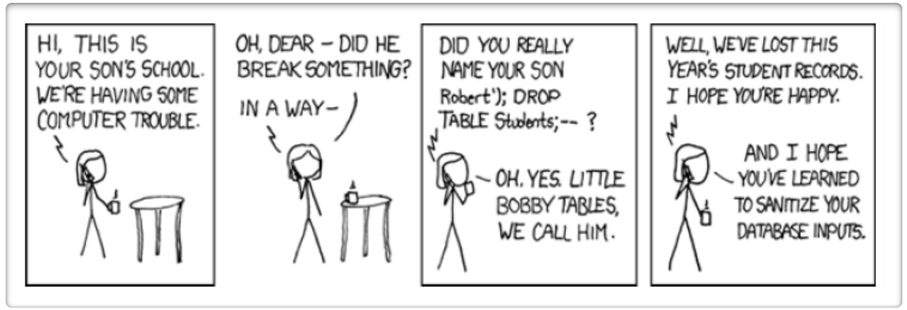
User: kihong, Password: 1234

```
query = "SELECT * FROM pswdtab WHERE username='kihong' AND password='1234'"
```

User: root' --, Password: 1234

```
query = "SELECT * FROM pswdtab WHERE username='root' -- AND password='1234'"
```

Funny


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

ZU 0666',0,0); DROP DATABASE TABLICE;--

'OR 1=1;--

How to Prevent?

- Secure coding

Don't use user input as code

```
str = input();
mysql_query(str);
```

```
str = substr(input(), 0, 8);
mysql_query(str + " * FROM tbl");
```

Sanitize user input (e.g., < instead of <)

Use parameterized queries

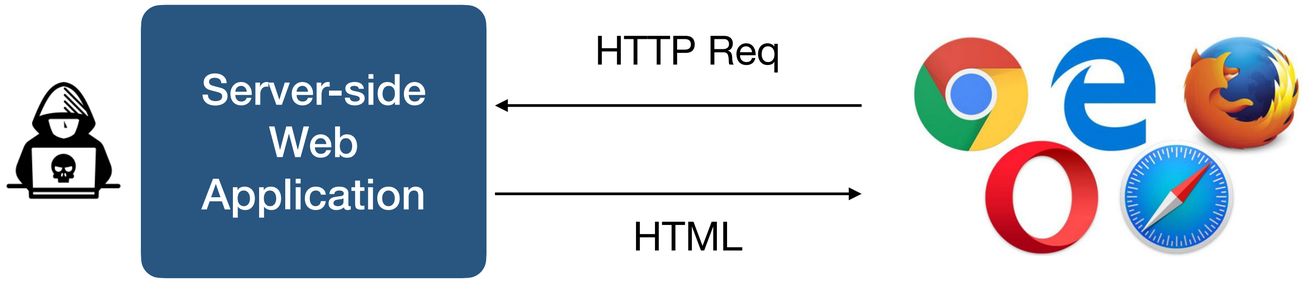
```
String userSuppliedParam =
    request.getParameter("Product-Description");
List<Inventory> list =
   session.getNamedQuery("findByDescription")
    .setParameter("productDescription", userSuppliedParam)
    .list();
```

- Automated program analysis
  - Statically or dynamically check the code before it is shipped

Cross-site Script

Simple Script Injection Attack

- Injection of malicious scripts into web pages
- Naive example: send a victim's cookies to an attacker site




```
Hello World!
<script>
  // send all private data
  // to malicious.com
</script>
```

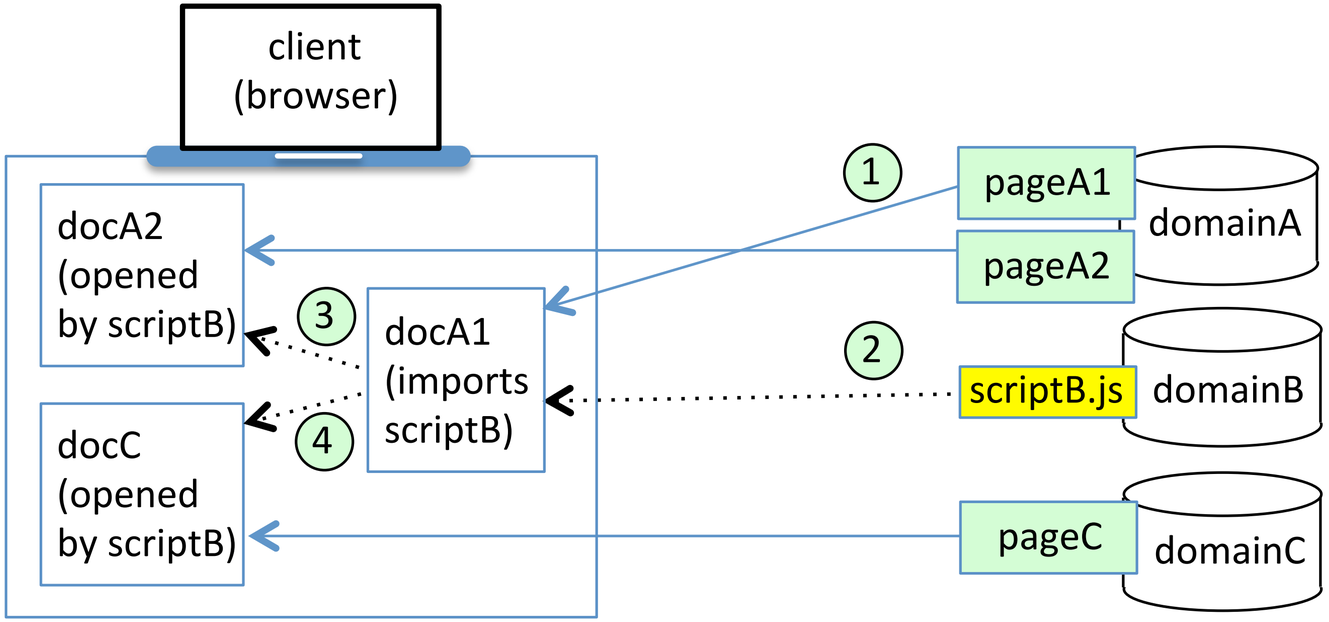
Contents from Many Sources

- One HTML document = multiple contents from distinct sources
  - Scripts: `<script src="dom1.com/script.js">`
  - Frames: `<iframe src="dom2.com/frame.html>`
  - Stylesheets (CSS):
    ```
    <link rel="stylesheet" type="text/css"
     href="dom3.com/theme.css">
    ```
- What if a script from `malicious.com` can access data from `bank.com`?

Same-origin Policy (SOP)

- Basic access control mechanism for web browsers
- Idea: allow a subject to access only the objects from the same origin
- Origin
  - For all resources and scripts such as DOM, cookies, JavaScript, etc
  - Scripts and images are assigned the origins of the host HTML documents
  - E.g., A script from `malicious.com` cannot access data from `bank.com`
- Designated by <protocol, host, port>

Example




```
// scriptB.js
1.  load pageA2 into docA2
2.  load pageC into docC
3.  access docA2
4.  access docC
```

Q1: Can scriptB access docA2?
Q2: Can scriptB access docC?

Does SOP solve all the problems?

*Figure from Oorschot's book

Cross-site Script (XSS)

- Injection of **malicious scripts** into otherwise benign websites
- **Bypass SOP** by making the pages from benign websites run malicious scripts
  - E.g., script origin: bank.com, data origin: bank.com
- How to bypass?
  - Reflected XSS: pass a malicious script in a parameter of a URL (e.g., query)
  - Stored XSS: store a malicious script in the victim server (e.g., bank)
  - Etc

Reflected XSS (1)

- Victim (vulnerable) web server, say www.victim.com

Server

```
<?PHP
    echo “Welcome! ” . “$_GET[‘user’].”;
?>
```

User

www.victim.com/index.php?user=Kihong

Client

```
<!DOCTYPE html>
<html>
<head>
<title>Victim</title>
</head>
<body>
   Welcome! Kihong
</body>
</html>
```

Browser

Reflection?

Reflected XSS (2)

- Victim (vulnerable) web server, say www.victim.com

Server

```
<?PHP
    echo “Welcome! ” . “$_GET[‘user’].”;
?>
```

User

```
www.victim.com/index.php?user=Kihong<script>alert(‘xss’)</script>
```

Client

```
<!DOCTYPE html>
<html>
<head>
<title>Victim</title>
</head>
<body>
   Welcome! Kihong
   <script>alert(‘xss’)</script>
</body>
</html>
```

Browser

Reflected XSS (3)

- Entice a user to visit the victim web server with specially crafted queries
  - Using phishing mails, attacker's web sites, etc

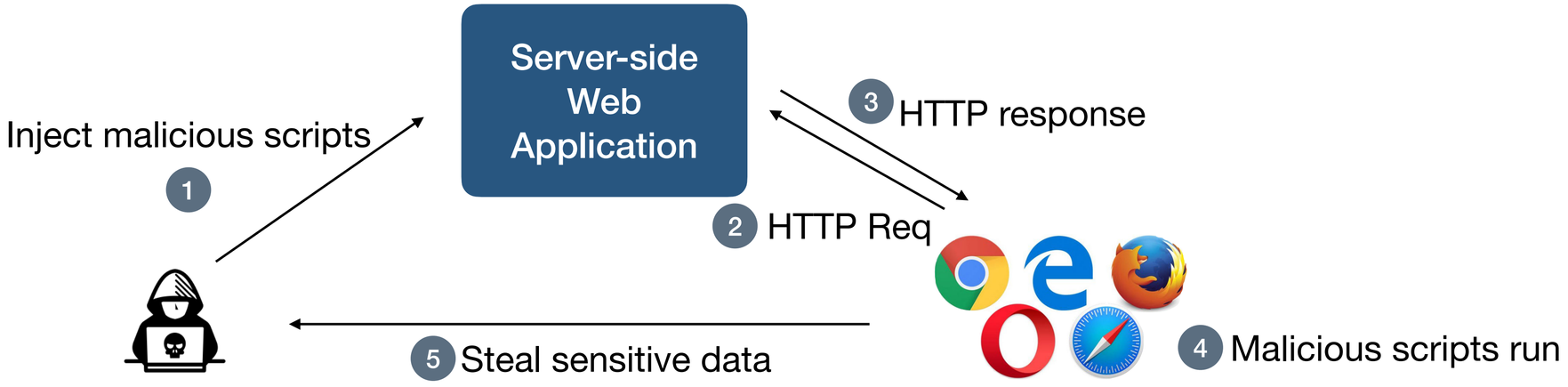
    "Check this out: http://www.victim.com?user=<script>…something bad…</script>"

- Malicious scripts will run on the user's machine (origin = victim)
  - E.g., steal sensitive data from bank, medical record, email, etc

Stored XSS Example

- Consider a victim (vulnerable) web forum
- An attacker writes a post that embeds a malicious script
- Malicious scripts will run on the user's machine (origin = victim)

```
Good morning!
<script>
  // Send all private data to malicious.com
</script>
```



Real World Examples

XSS On Twitter [Worth 1120$]

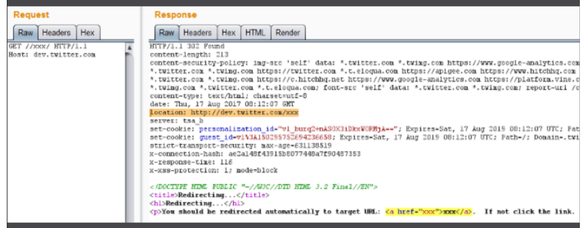
Bywalks

Hi guys, this is the first writeup about my vulnerability bounty program,a process about how I discovered a Twitter XSS vulnerability.

I think that in the process of finding the vulnerability, there are some interesting knowledge points, I hope you can get some from my writeup.

*If you want to know more details, you need to visit **bobrov's blog**, my discovery is due to reading his writeup, and thanks bobrov very much,I have a lot of gains from his blog.*

Maybe you don't want to spend more time. Here I will give a brief explanation of his article. When you visit some addresses, the server returns 302, which is similar to the following picture.



In the returned Body, location will choose how to populate according to the requested URL,and the requested URI will be placed in the href event.

What do you think of next? Can we try it with ***dev.twiiter.com//javascript:alert('1');/***

Stored XSS bug in Apple iCloud domain disclosed by bug bounty hunter

The cross-site scripting bug reportedly earned the researcher a $5000 reward.

Charlie Osborne • February 22, 2021 -- 12:03 GMT (20:03 SGT)

A stored cross-site scripting (XSS) vulnerability in the iCloud domain has reportedly been patched by Apple.

Bug bounty hunter and penetration tester Vishal Bharad claims to have discovered the security flaw, which is a stored XSS issue in icloud.com.

Stored XSS vulnerabilities, also known as persistent XSS, can be used to store payloads on a target server, inject malicious scripts into websites, and potentially be used to steal cookies, session tokens, and browser data.

According to Bharad, the XSS flaw in icloud.com was found in the Page/Keynotes features of Apple's iCloud domain.

In order to trigger the bug, an attacker needed to create new Pages or Keynote content with an XSS payload submitted into the name field.

This content would then need to be saved and either sent or shared with another user. An attacker would then be required to make a change or two to the malicious content, save it again, and then visit "Settings" and "Browser All Versions."

After clicking on this option, the XSS payload would trigger, the researcher said.

Bharad also provided a Proof-of-Concept (PoC) video to demonstrate the vulnerability.

How to Prevent XSS?

- Input sanitization: remove potentially malicious elements from data
  - E.g., delete tags and event attributes such as <script>, <embed>, etc
- Content security policy (CSP): servers declare trusted sources
  - A new security mechanism supported by modern browsers
  - CSP: server-side, SOP: client-side
  - E.g., disallow all inline scripts
  - https://content-security-policy.com

Content Security Policy

- Policy is specified in server's response
- A list of pairs of a directive and values
- Example directives
  - default-src: default policy for loading contents
  - script-src: define valid sources of JavaScript
  - img-src: define valid sources of images
- Example values: 'self', kaist.ac.kr

Example

- Example:

default-src ‘none’; script-src ‘self’; img-src ‘self’; object-src ‘self’

load scripts, images, and objects from the same origin
but do not load anything

default-src ‘none’; script-src ‘self’ kaist.ac.kr

load scripts only from the same origin and kaist.ac.kr

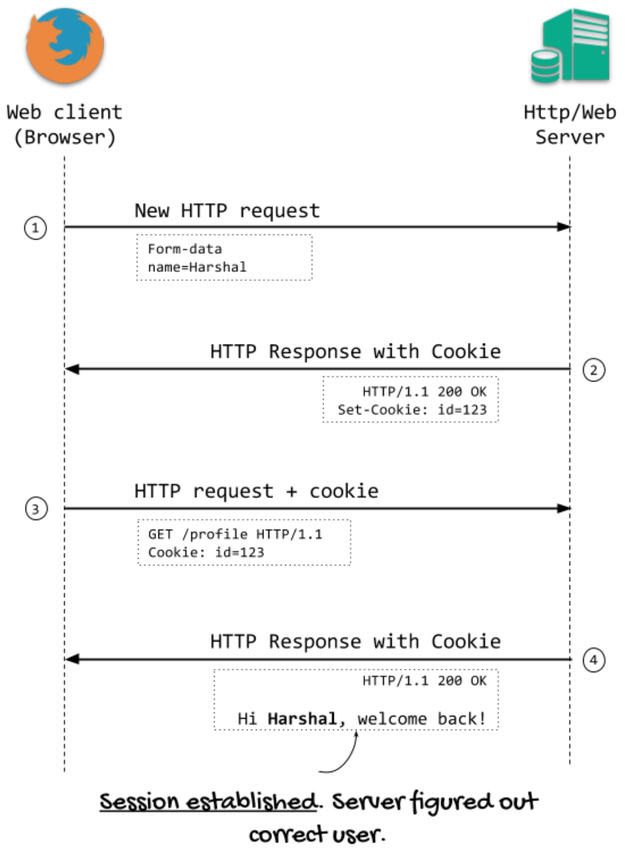
script-src ‘unsafe-inline’

load anything including even inline scripts (Be careful!)

Cross-site Request Forgery (CSRF)

Recall: Cookies

- A common usage: authentication
  - E.g., log into bank.com
- Once authenticated, subsequent requests will be accepted
- What if an attacker **tricks the user** to do unwanted actions?
  - E.g., send money to the attacker



Cross-site Request Forgery (CSRF)

- Force a user to execute unwanted actions on an **authenticated** web application
- Trick users with **social engineering** such as
  - URL: <a href="bank.com/transfer.php?to=badguy&amount=100>View my Pictures!</a>
  - Invisible image: <img src="bank.com/transfer.php?to=badguy&amount=100 height=0 width=0>
- SOP violation?

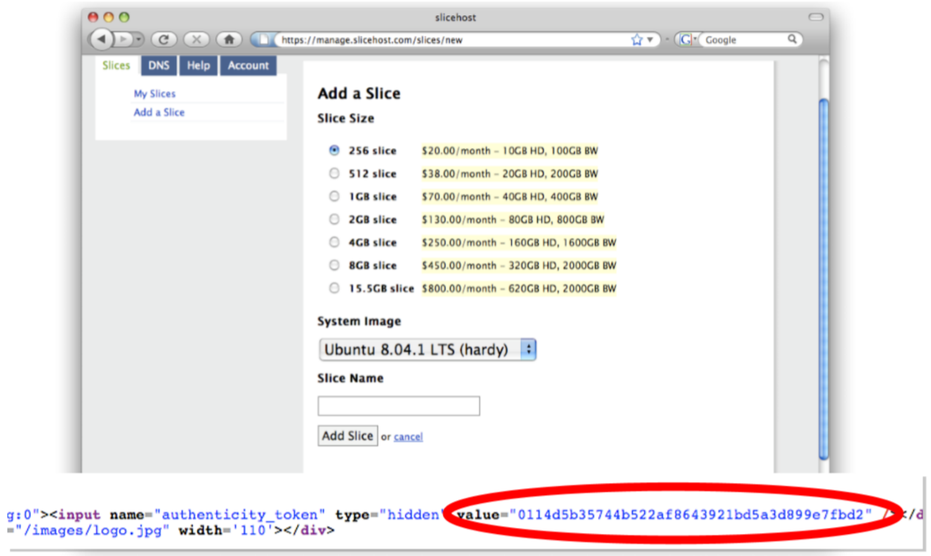
How to Prevent CSRF?

- Referrer checking: “where is this request coming from?”
  - Accept requests only if their referrer is the same as the server (e.g., *.bank.com)
- Secret validation token
  - For each session, a fresh secret token is generated by the server
  - Send requests with the token
  - Accept requests only if the token is valid

Secret Validation Token

```
// Fake submission page
<form name=BillPayForm action="bank.com/transfer.php>
<input name=recipient value=badguy>
<script>document.BillPayForm.submit(); </script>

// Valid submission page
<form name=BillPayForm action="bank.com/transfer.php>
<input name=recipient value=friend>
<input type=hidden name="csrf-token" value="Jk4j392tjfaijwe">
<script>document.BillPayForm.submit(); </script>
```



Summary

- Web security: safety of browser-server interactions
- Same origin policy: basic access control
- Common vulnerabilities:

	Attack Target	Attack Method	Mitigation
SQL Injection	Server	Malicious query	Sanitization Program analysis
XSS	Server / Client	Malicious script	Sanitization, CSP, Program analysis
CSRF	Server / Client	Fake request	Referrer checking, Secret validation token Program analysis