Web Security

with slides from Karen and Irving Reid

This stuff matters.

Getting hacked is no fun.

- Wastes your time cleaning up
- Reputation damage to you, your employer, your users
- Financial damage to the same list
- Stolen information can be used in attacks elsewhere
- Your site could end up hosting malware

You won’t escape.

- People are constantly scanning for vulnerable servers
- Looking for unpatched systems, trying hacks
- Some exploits fully automated, others followed up by humans

Goals

Confidentiality

- the correct people have access to information

Integrity

- the information is reliable

Availability

- users can accomplish their tasks

Risks are bad things that could happen:

- Financial
- Reputation
- Physical harm

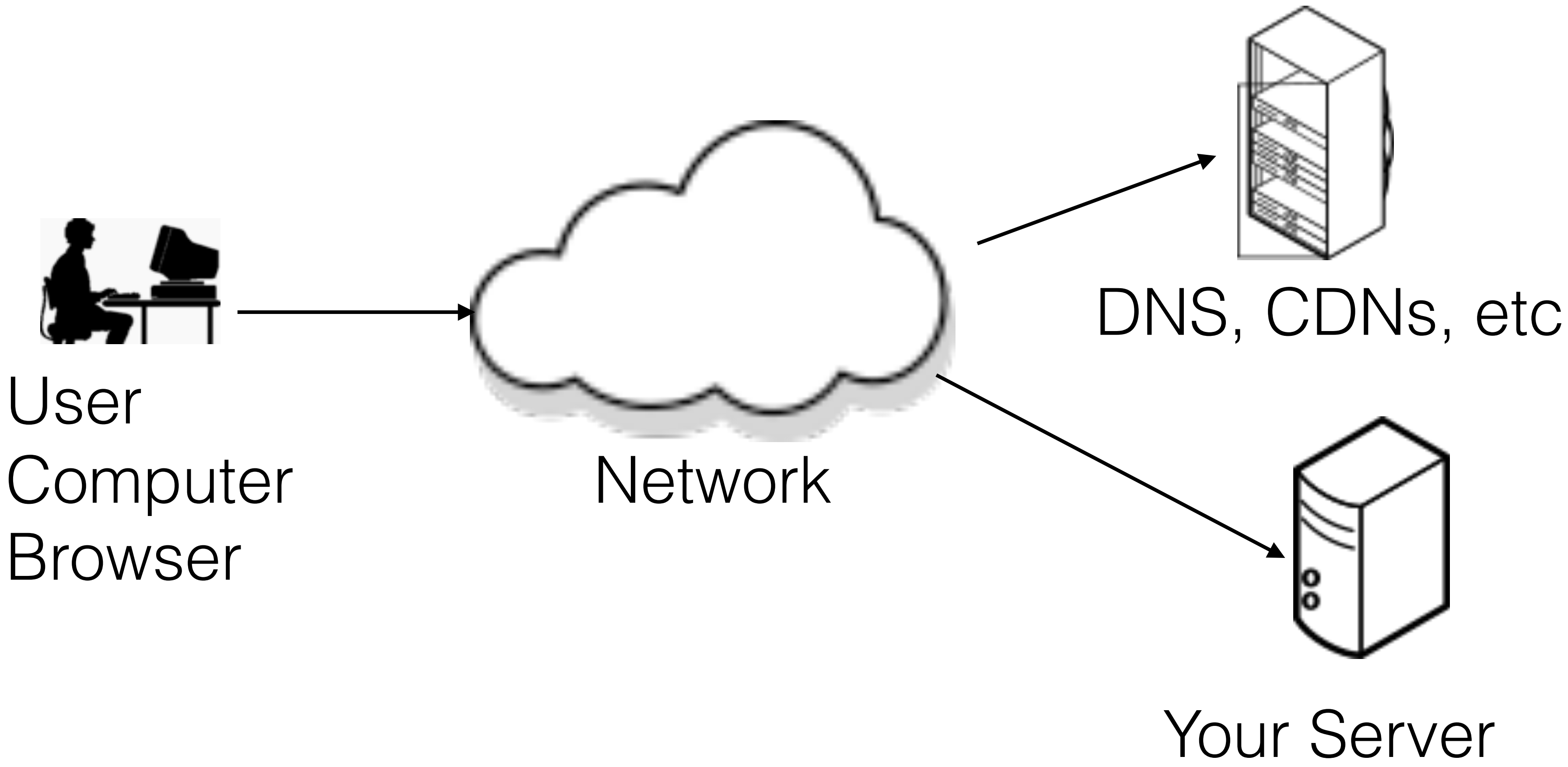
Threats are potential causes of risk:

- Insiders
- Criminals
- Commercial competitors
- Nation-states (intelligence agencies and their proxies)
- Law Enforcement
- Vandals, “security researchers”, “script kiddies”

All sort of attacks

- Directly on your system
  - stealing data, passwords, credit card numbers
  - defacing, denial of service, link spam
- On your users
  - Cross Site Scripting (XSS), Request Forgery (CSRF), Man-In-The-Middle (MITM), profiling
- Both
  - Hosting bad content / “drive-by download”
- Neither
  - Ad-based malware

Environment



(https://openclipart.org/)

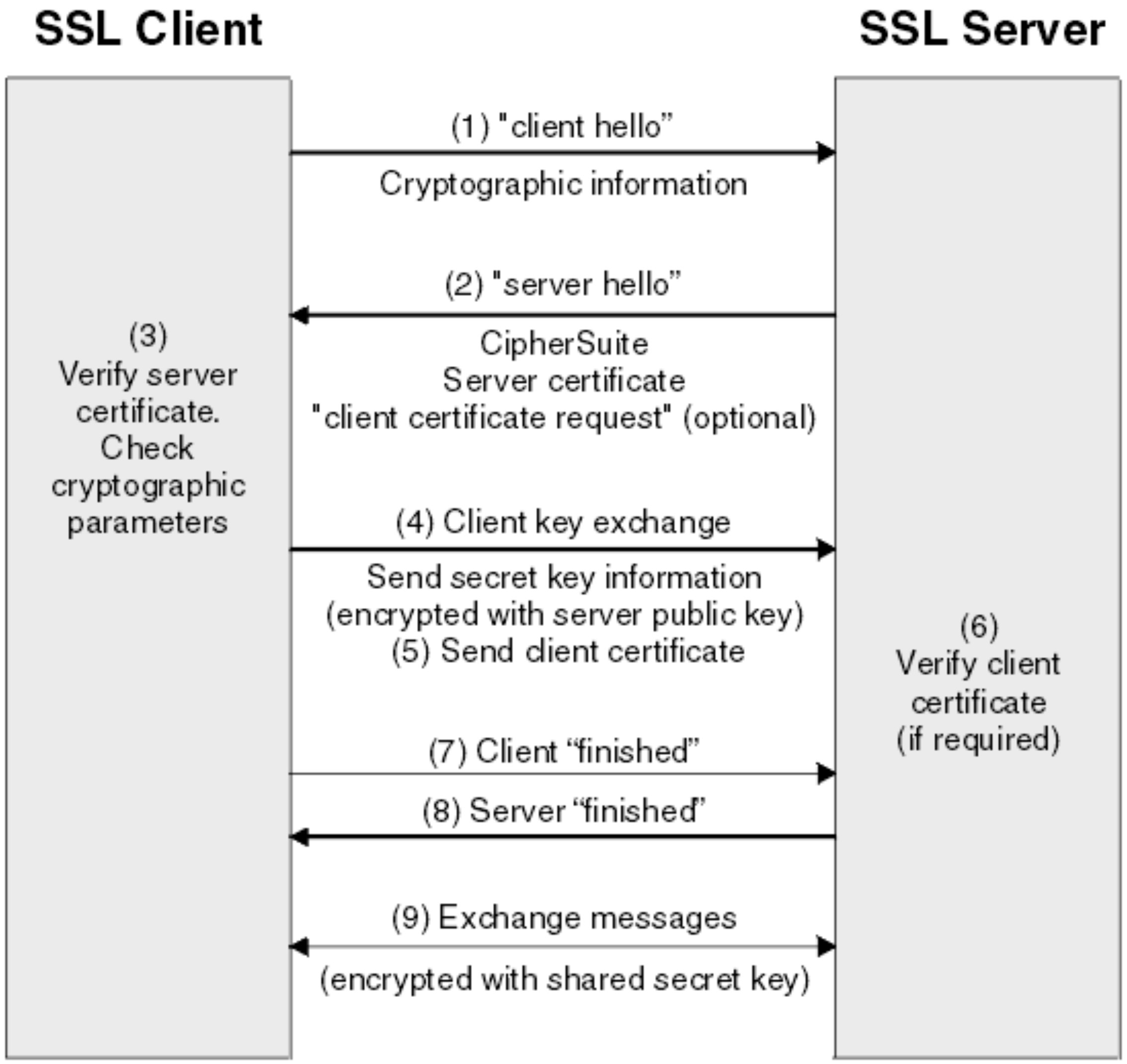
Transport Layer Security (TLS)

Security mechanism underlying HTTPS

Often still called SSL, but the older SSL protocol versions are obsolete and broken

Client and Server use public-key encryption to agree on a shared per-session secret, then use that secret to encrypt session data.

http://www.ibm.com/support/knowledgecenter/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm


SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

TLS and You

- You owe it to your users
- Get a server certificate from Let’s Encrypt
  https://letsencrypt.org/
- Test your configuration, e.g.
  https://www.ssllabs.com/ssltest/

Authentication

- Who is the user?
- Don’t write your own
- Always store passwords salted & hashed, using trusted algorithms (PBKDF2, scrypt, bcrypt - see https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet)

“Social” login

- Facebook Login, Google Identity, Sign In With Twitter, etc.
- OAuth 2.0 (https://oauth.net/2/)
- Federated Identity
- SAML (https://wiki.oasis-open.org/security/FrontPage#SAML_V2.0_Standard)

Authorization

We know who you are, what can you do?

- No silver bullet
- Look for support in your web framework
- Check every operation

Open Web Application Security Project

- https://www.owasp.org/index.php/About_OWASP

- Not for profit, registered charity in US and Europe

- Unbiased advice about common web security flaws and how to address them

- Top 10 last updated 2013, but things haven't changed much

- https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013

Learn and Practice

Lists of security teaching web sites you can explore and try to hack:

- https://www.checkmarx.com/2015/04/16/15-vulnerable-sites-to-legally-practice-your-hacking-skills/
- https://hack.me/

OWASP Top 10

1. Injection
2. Broken Authentication and Session Management
3. Cross Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing function level access control
8. Cross Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

1. Injection

- SQL Injection
  - User inputs data that you want to store in the database
  - You insert that data into an SQL query
  - E.g.

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

```
                foo' or 'x'='x
```

```
String query = "SELECT * FROM accounts WHERE
custID=' foo' or 'x'='x '";
```

- returns all customer information

Example

- Suppose we want to identify a user by their email address.
- Input field named “email”
- Using the same techniques as above, we can figure out parts of the database schema, and make an educated guess about the SQL query and the email address of a victim.

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x';
      UPDATE members
      SET email = 'steve@unixwiz.net'
      WHERE email = ‘bob@example.com';
```

- Now we can follow the normal password reset and hijack Bob’s account

Real shell example

- Instructor writes shell script to checkout student repositories (**`checkoutrepos.sh`**)
- Each checkout needs instructor's password
- Instructor does not want to save his password in plaintext in script in his account, so he writes the script to take the password as command line argument.

```
$ ./checkoutrepos.sh fakepassword
$ ps aux | grep reid
reid@wolf:~$ ps aux |grep reid
```

Anyone with an account can run this.

```
reid      50029  0.0  0.0  22124  7280 pts/228  Ss   23:33   0:00 -bash
reid      50301  0.0  0.0   9520  2412 pts/227  S+   23:36   0:00 /bin/bash
./checkoutrepos.sh fakepassword
reid      50302  0.0  0.0   4348   640 pts/227  S+   23:36   0:00 sleep 120
reid      50325  0.0  0.0  15568  2252 pts/228  R+   23:36   0:00 ps aux
reid      50326  0.0  0.0   8868   768 pts/228  S+   23:36   0:00 grep reid
```

Injection Prevention

- Use parameterized queries
- Escape everything
- Validate input
- Never trust raw input

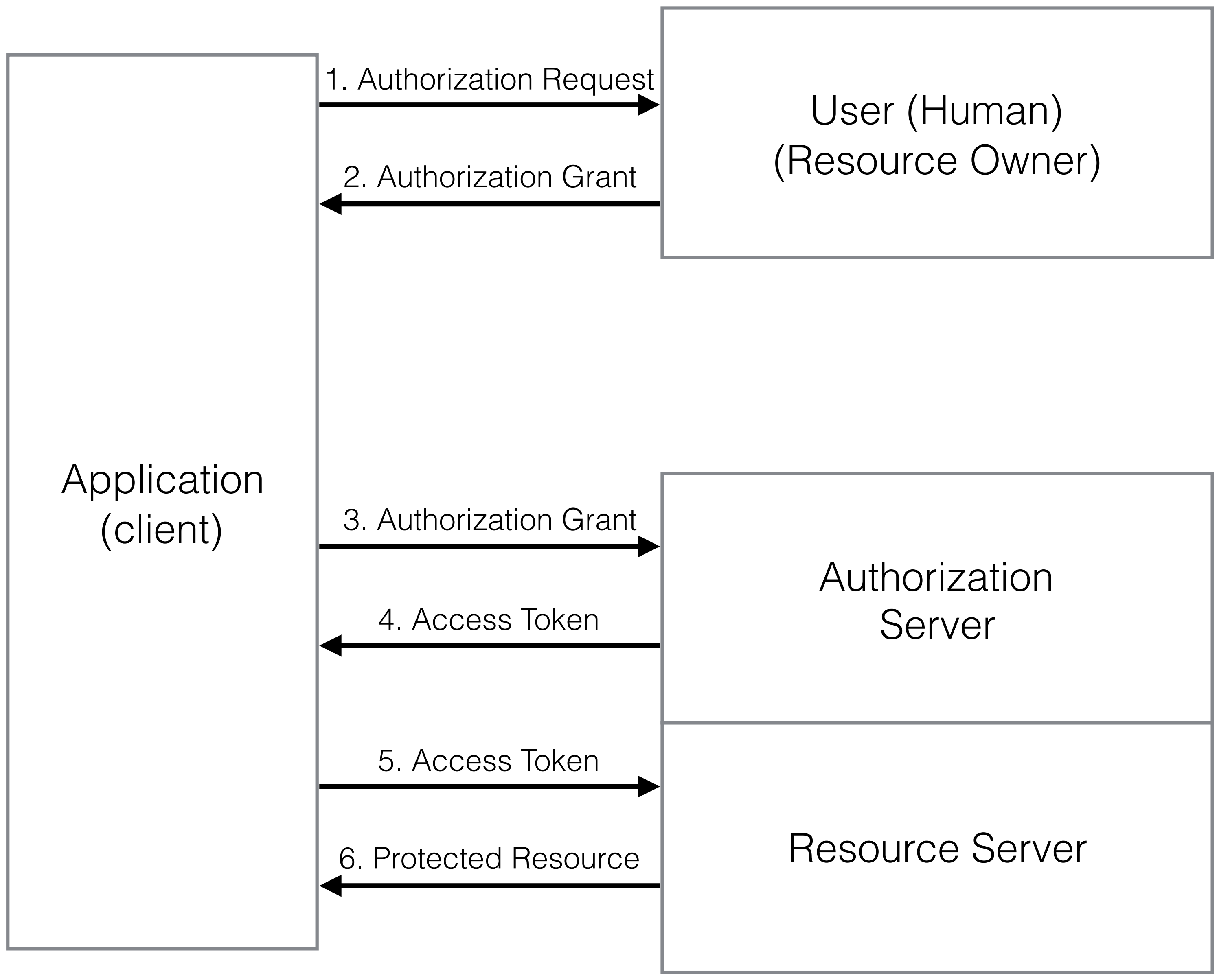
2. Auth and Sessions

- Vulnerabilities
    - User credentials not stored using hashing or encryption
    - Credentials can be guessed or overwritten
    - Session IDs exposed in URL
    - Session IDs don’t time out
    - Authentication tokens (single sign-on) not properly invalidated on log out
    - Passwords, session IDs sent over unencrypted connections

OAuth

- Authorization framework that delegates user authentication to a service that hosts the user account
- Must register app with the auth service, including the URL to redirect to
- Auth service redirects to a registered URL with an authorization code (using HTTPS of course)

Basic idea


Application (client)
1. Authorization Request
2. Authorization Grant
User (Human) (Resource Owner)
3. Authorization Grant
4. Access Token
Authorization Server
5. Access Token
6. Protected Resource
Resource Server

3. Cross-Site Scripting

- Data enters web application and is included in dynamic content sent to a web user without being validated for malicious content.
- Usually JavaScript but may also include HTML and other data
- Three types
  - Stored
    - injected script is stored permanently on server
    - victim retrieves script from server through normal requests
  - Reflected
  - DOM-based
- Escape all untrusted data

4. Insecure Direct Object Reference

- Attacker is an authorized user
- Attacker can gain access to objects by guessing parameters

```
http://example.com/app/accountInfo?
acct=notmyacct
```

- Prevention: ensure user is authorized for access

5. Security misconfiguration

1. Is any of your software **out of date**? This includes the OS, Web/App Server, DBMS, applications, and all code libraries (see new A9).

2. Are any unnecessary features enabled or installed (e.g., ports, services, pages, accounts, privileges)?

3. Are **default accounts** and their **passwords** still enabled and unchanged?

4. Does your error handling reveal **stack traces** or other overly informative error messages to users?

5. Are the **security settings** in your development frameworks (e.g., Struts, Spring, ASP.NET) and libraries not set to secure values?

6. *Without a concerted, repeatable application security configuration process, systems are at a higher risk.*

6. Sensitive Data Exposure

- Which data is sensitive enough to require extra protection?
  - passwords
  - credit cards
  - personal information
- Is any of this data ever transmitted in clear text?
- Encrypt all sensitive data

Examples

- Example 1:
  - Automatic database encryption of passwords
  - This means it is also automatically decrypted
  - Passwords still vulnerable to SQL injection attack
- Example 2:
  - Application doesn’t use TLS for all authenticated pages
  - Network packet snooping can read session cookies
  - Replay session cookies to hijack users’s session

7. Missing Function Level Access Control

- Modifies URL, changes parameter and gets access to data attacker is not authorized for

- Often URL in frameworks refers directly to objects
    - Internal ids appear in URLs (REST)

- Check access authorization on every object

- Not sufficient to simply not show privileged operations to unprivileged users in the UI

8. Cross-Site Request Forgery

- Some content on unrelated site includes a POST to your application
- If a user of your app navigates to compromised site while logged into your app, the malicious POST request can pretend to be the user, and steal the user’s info from your app
- Prevention: Include an unpredictable token with each HTTP request (usually in a hidden field)

9. Using components with know security vulnerabilities

- Development teams and Dev Ops people must be vigilant
- Be careful what you include in your application
- Update software always

10 Unvalidated Redirects and Forwards

- Apps use redirects or internal forwards where unvalidated parameters are used
- Example

```
http://www.example.com/redirect.jsp?url=evil.com
```