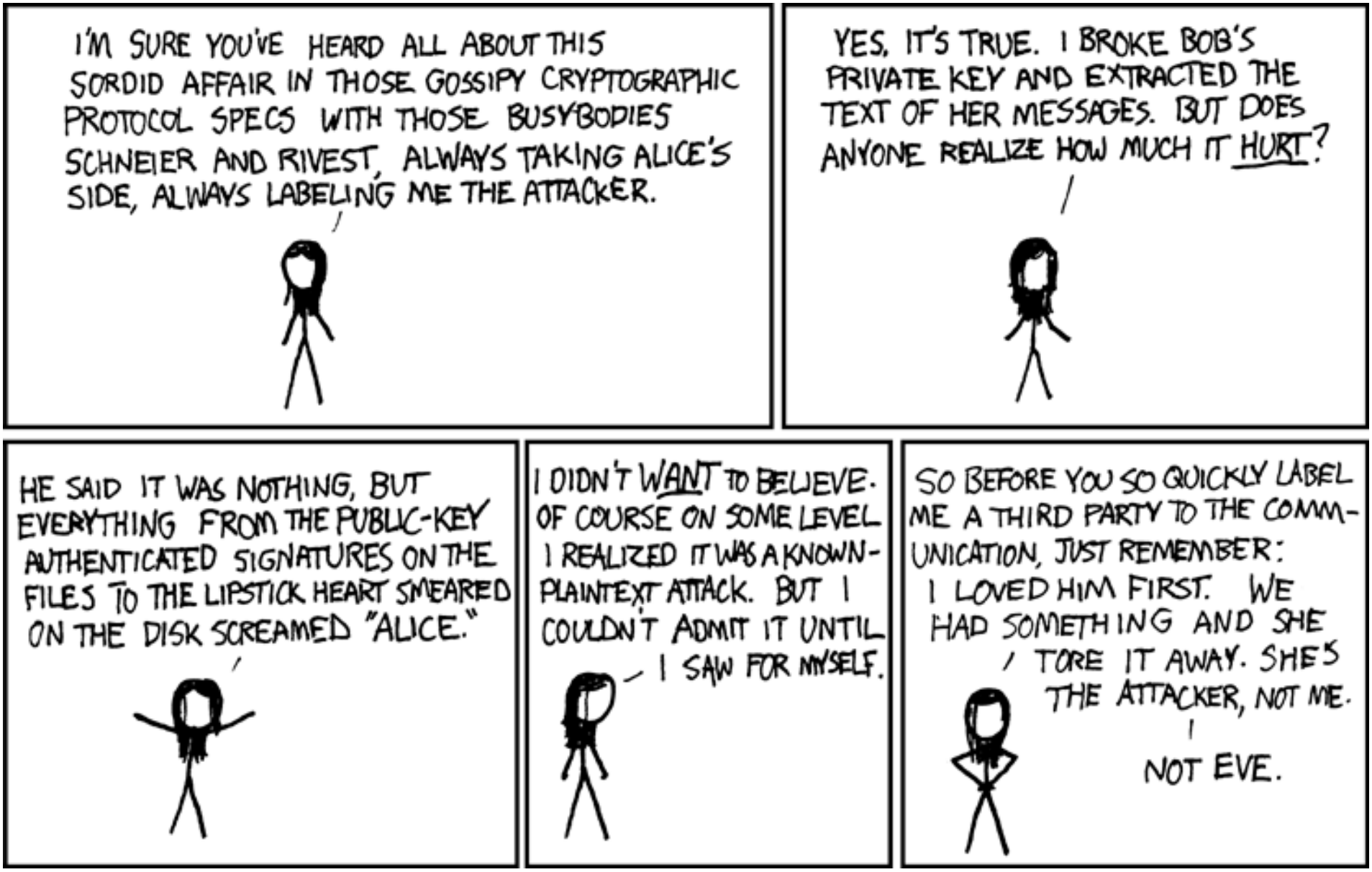
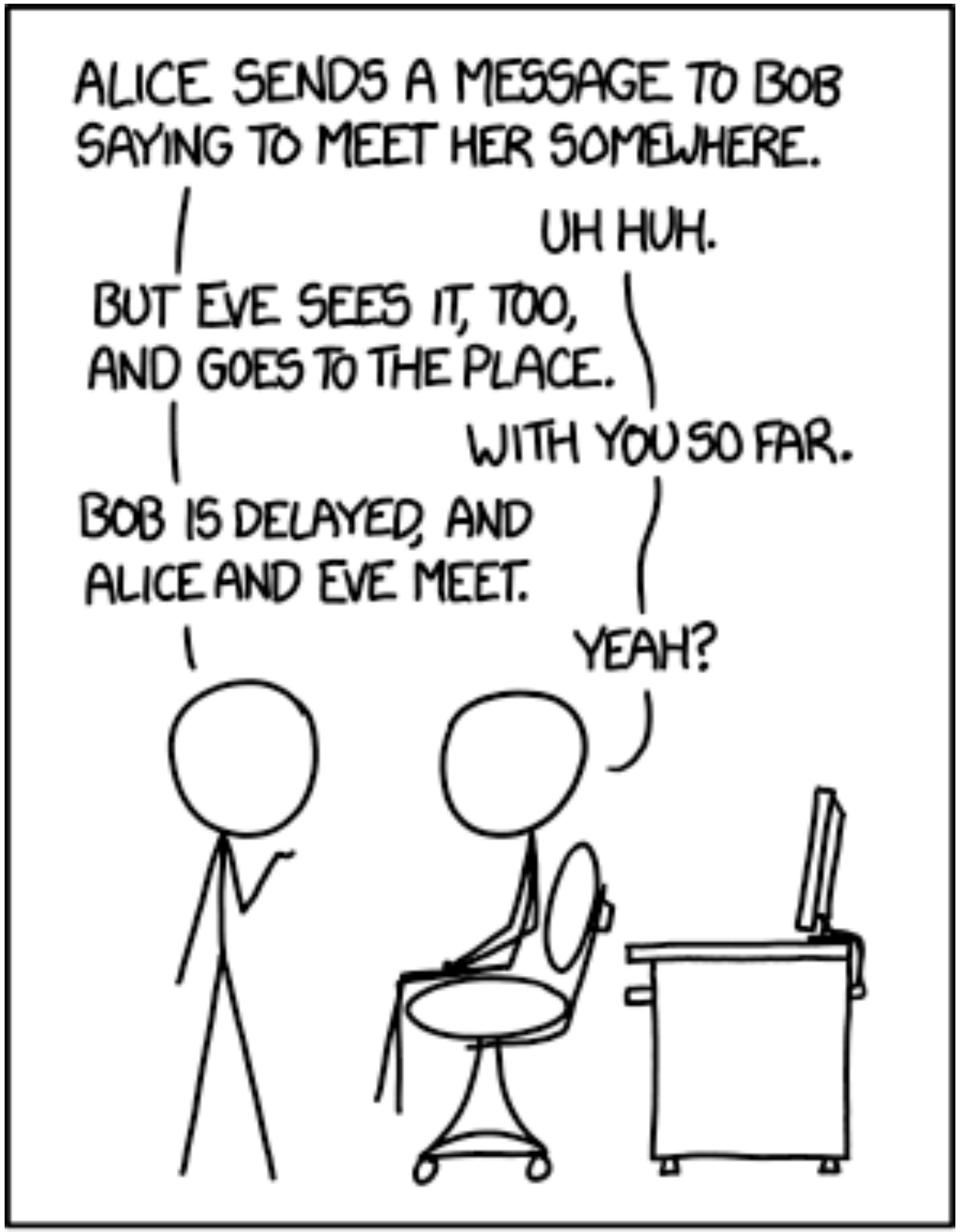
Web Security 2

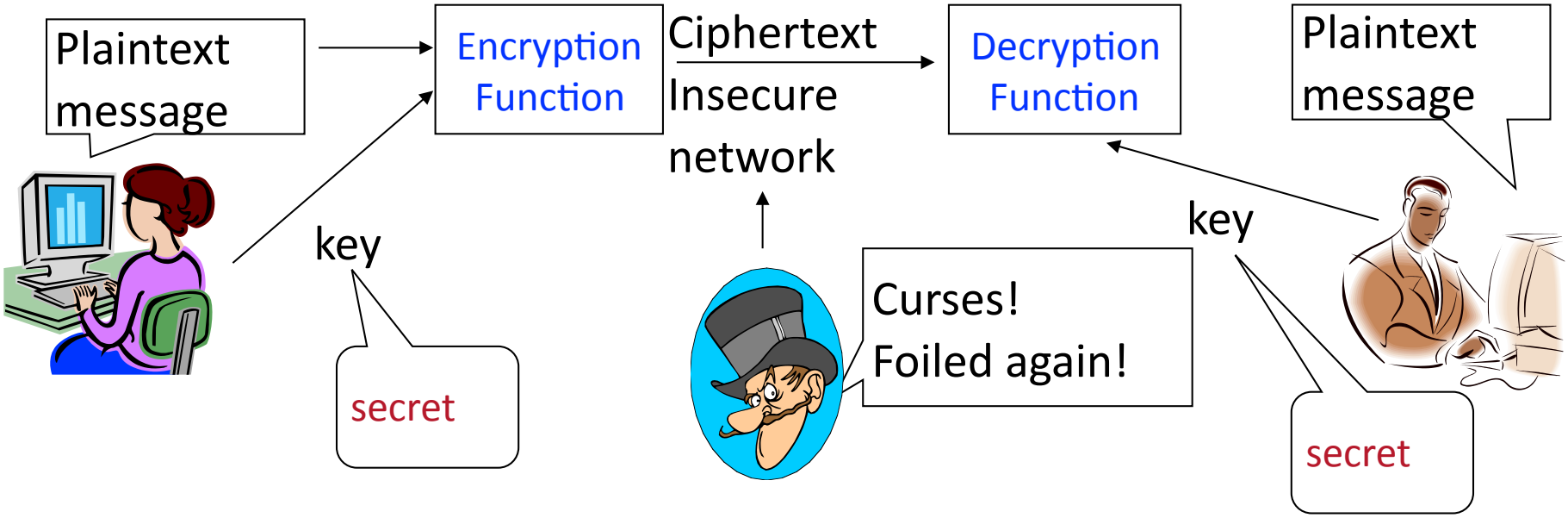


https://www.xkcd.com/177/



http://xkcd.com/1323/

Encryption basics



- Keys used for encryption and decryption may be same (symmetric) or different (public key)

Diffie Hellman Merkle Key Exchange

- Function: g modulo p (where p is prime)
- Alice and Bob publicly agree to use
  - p = 23, g = 5
- Alice chooses secret 6
  - sends Bob 5^6 mod 23 = 8
- Bob chooses secret 15
  - sends Alice 5^{15} mod 23 = 19
- Alice computes s = 19^6 mod 23 = **2**
- Bob computes s = 8^{15} mod 23 = **2**

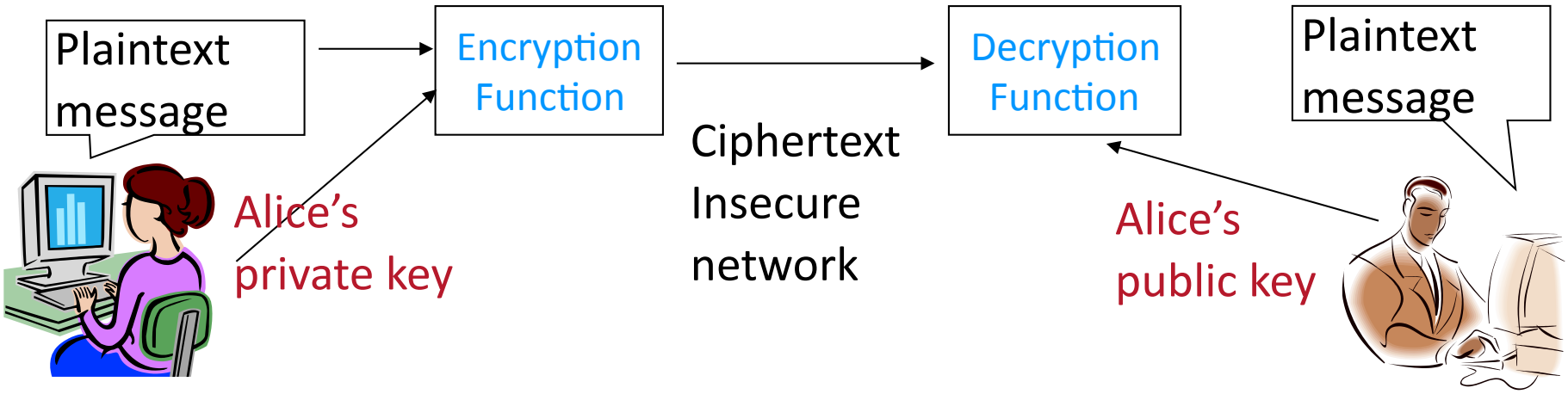
Eve can know p, g, 8, 19, and it is *hard* to compute 2

2 is the secret symmetric key

Encryption Keys

- Keys used for encryption and decryption can be the same (symmetric or secret-key) or different (public-key)
- Secret-key cryptography
  - Fast encryption/decryption algorithms are known
  - Distributing shared secret is a problem
- Public-key cryptography
  - Pairs of keys, one to encrypt, one to decrypt
  - Encryption key can be published, decryption key is kept secret; knowledge of one key does not reveal the other
  - Encryption/decryption algorithms ~1000x slower

Message signing



- Alice "signs" her message by appending a copy of her message encrypted with her private key.
- Bob uses the public key to decrypt the "signature" and can match the decrypted text to the plaintext, to make sure the text has not been tampered with.
- Bob can be sure that Alice sent the message.

SSH

- Establish TCP connection
- Agree on a session key
  - server sends public RSA key (host identity)
  - symmetric key exchange using Diffie-Hellman
  - client gets confirmation message from server encrypted with symmetric session key
- Authenticate client
  - password
  - Public key - client sends public key
    - server sends challenge encrypted with public key
    - client decrypts with private key
    - client sends challenge back encrypted with session key

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

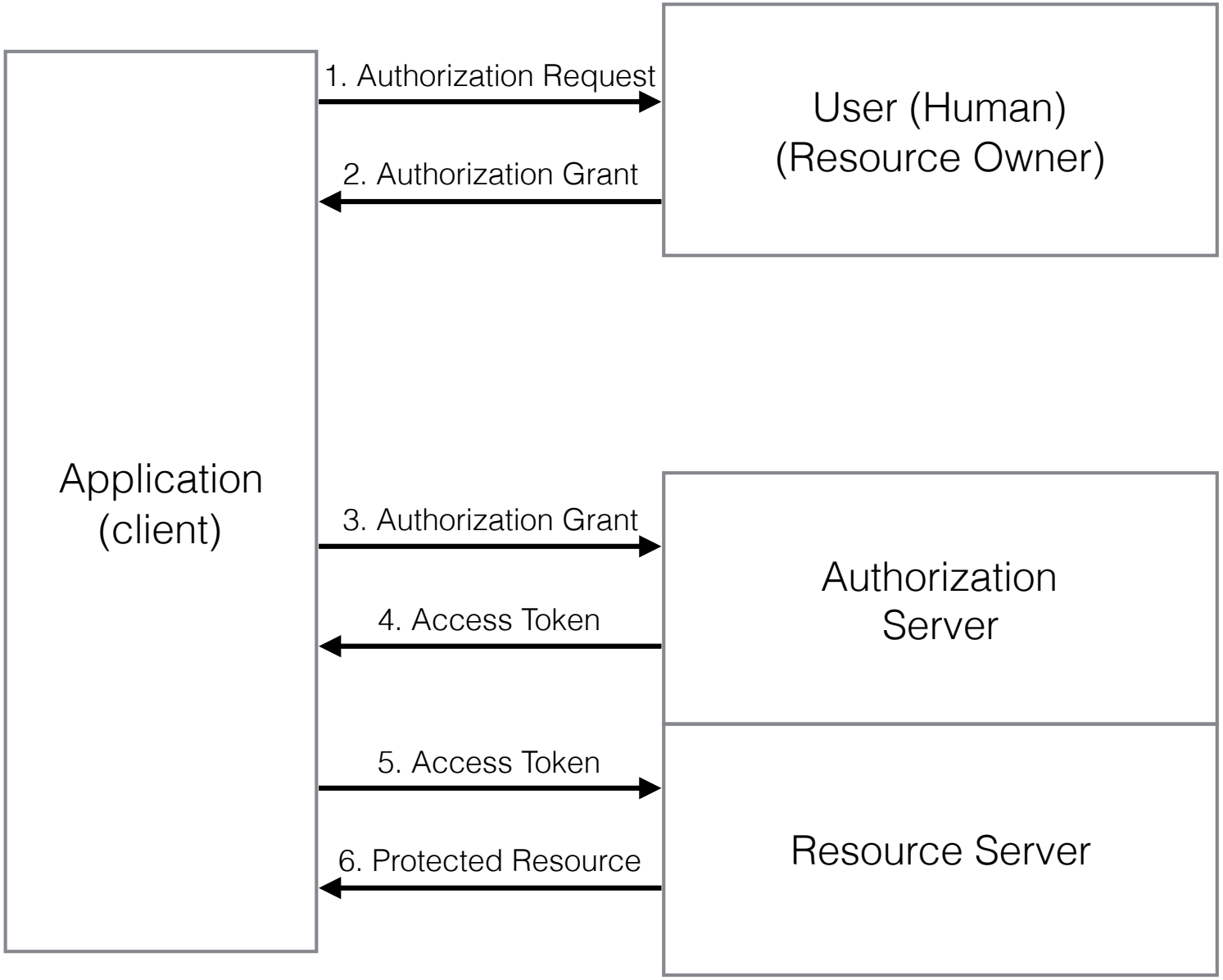
2. Auth and Sessions

- 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


1. Authorization Request
2. Authorization Grant
User (Human)
(Resource Owner)
Application
(client)
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
```

“Arguing that you don’t care about the right to privacy because you have nothing to hide is no different than saying you don’t care about free speech because you have nothing to say.”

–Edward Snowden

Go watch
“Citizen Four”

“If you think technology can solve your security problems, then you don’t understand the problems and you don’t understand the technology.”

–Bruce Schneier

“The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards.”

– Gene Spafford

Feeling good about the world?