Information Security
CS3002
(Sections BDS-7A/B)
Lecture 19

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science

23 October, 2024

Example: Employee Table (with encryption)

eid	ename	salary	addr	Did
23	Tom	70K	Maple	45
860	Mary	60K	Main	83
320	John	50K	River	50
875	Jerry	55K	Hopewell	92

Employee Table

E(k, B)	I(eid)	I(ename)	I(salary)	I(addr)	I(did)
1100110011001011…	1	10	3	7	4
0111000111001010…	5	7	2	7	8
1100010010001101…	2	5	1	9	5
0011010011111101…	5	5	2	4	9

Encrypted Employee Table with Indexes

Web Security

- Background
- Cross Site Request Forgery (CSRF) Attack
- Countermeasures (STP, origin header, referer header)

Web Security: Background

- *Website*
  - Collection of webpages
    - objects
  - HTML
  - CSS
  - JavaScript
- *Http/Https*
  - Request (GET, POST, HEAD, PUT, DELETE)
  - Reply (Codes i.e. 200, 400, 404)
- *Webservers*
  - Stateless
- *Cookies*

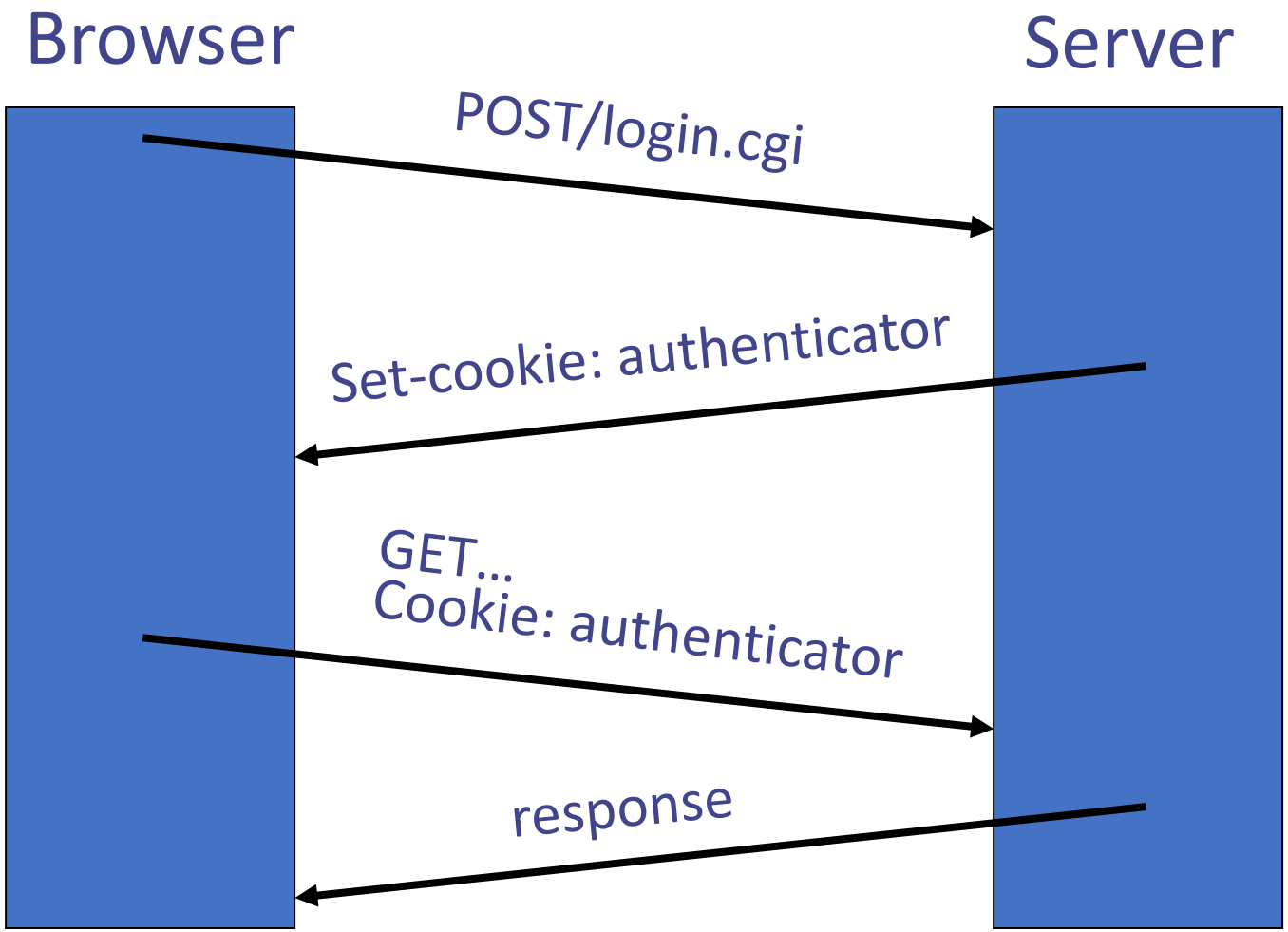
HTTP Request Message

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Cookie: 1678\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP Response Message

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Set-Cookie: 1678\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

Cookies are used for Session Management



CSRF: Cross Site Request Forgery (OWASP)

- ***Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.***

- With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into *executing actions of the attacker's choosing*.

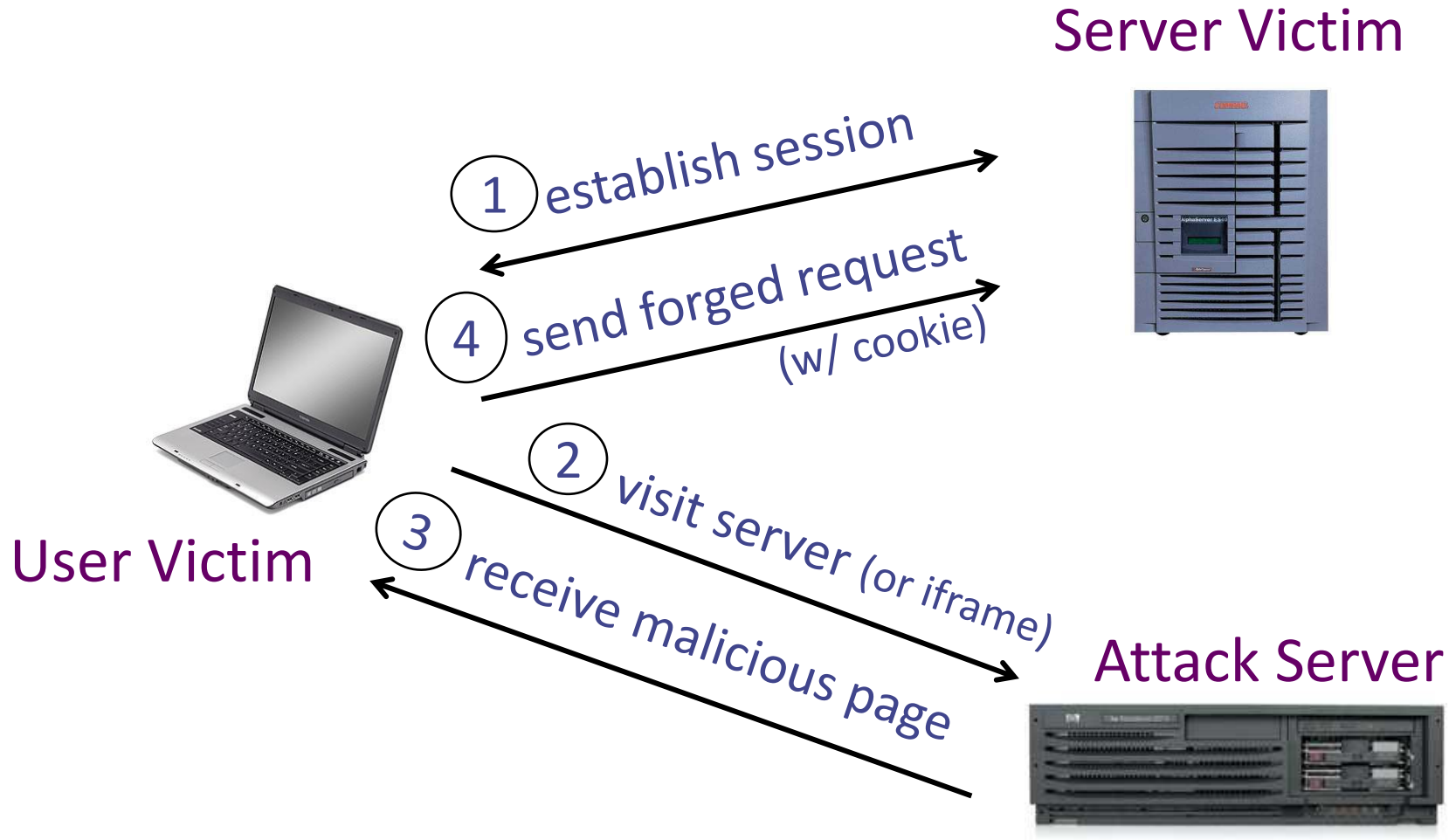
- If the victim is a normal user, a successful CSRF attack can force the user to perform *state changing requests* like transferring funds, changing their email address, and so forth.

- If the victim is an *administrative account, CSRF can compromise the entire web application*.

CSRF Attacks -*Impact and Effects*

- The impact of a successful CSRF attack is limited to the *capabilities exposed by the vulnerable application*.
  - For example, this attack could result in a *transfer of funds*, *changing a password*, or *purchasing an item in the user's context*.
- In effect, CSRF attacks are used by an attacker to *make a target system perform a function via the target's browser without knowledge of the target user*, at least until the unauthorized transaction has been committed.
- The victim's *privileges are important* here.

CSRF Attack



Q: how long do you stay logged in to Gmail? Facebook? ….

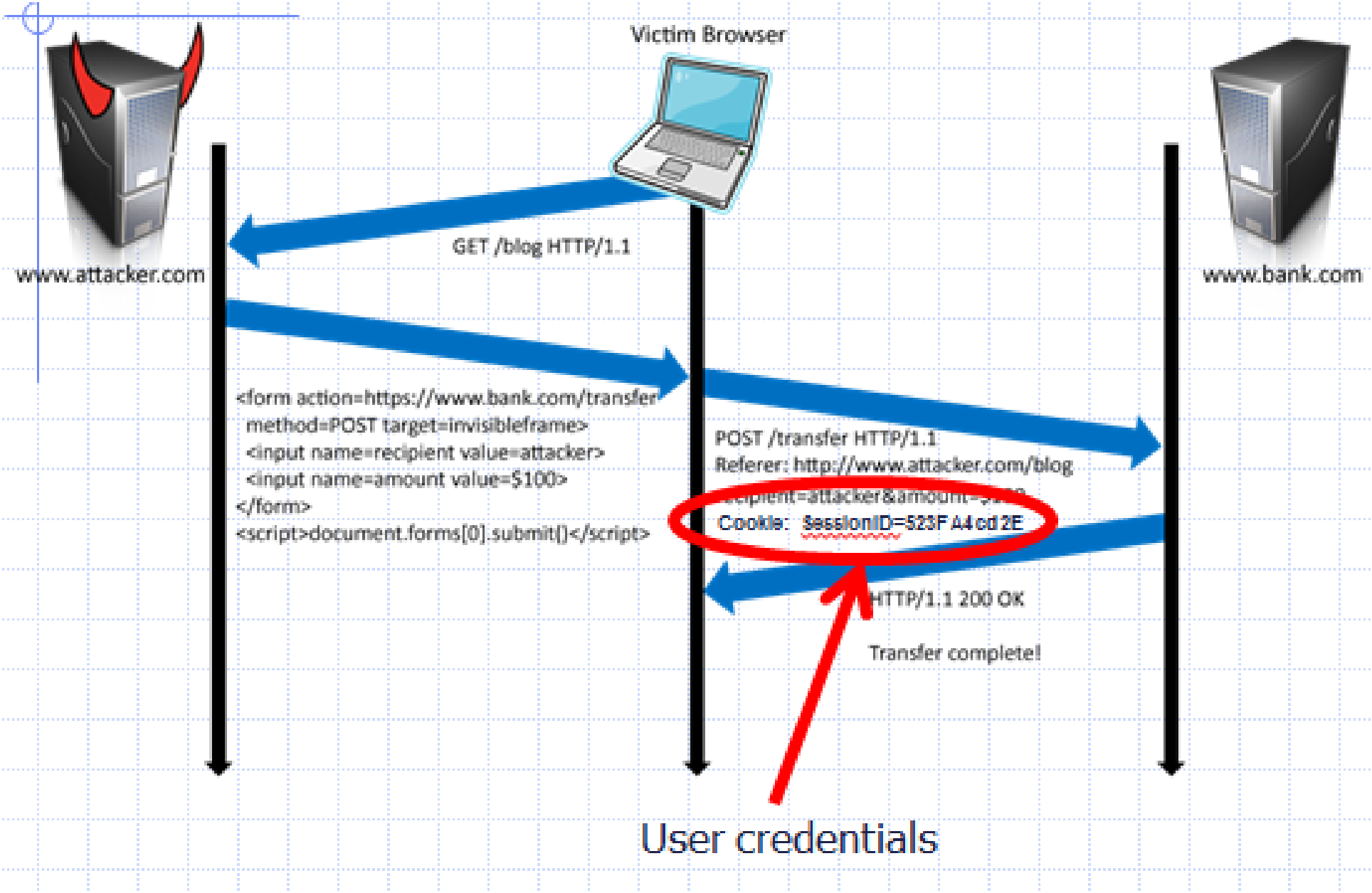
CSRF Attack

- Example:
  - User logs in to **bank.com**
    - Session cookie remains in browser state
  - User visits another site 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

CSRF Attack


Victim Browser
www.attacker.com
www.bank.com
GET /blog HTTP/1.1
<form action=https://www.bank.com/transfer
method=POST target=invisibleframe>
<input name=recipient value=attacker>
<input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
Cookie: SessionID=523FA4cd2E
HTTP/1.1 200 OK
Transfer complete!
User credentials

Real Life Scenarios

- CSRF vulnerabilities have been known and in some cases exploited since 2001. Because it is carried out from the *user's IP address*, some website logs might not have evidence of CSRF.
- Exploits are under-reported, at least publicly, and as of 2007 there are few well-documented examples:

  - The **Netflix** website in 2006 had numerous vulnerabilities to CSRF, which could have allowed an attacker to perform actions such as adding a DVD to the victim's rental queue, changing the shipping address on the account, or altering the victim's login credentials to fully compromise the account.
  - The online banking web application of **ING Direct** was vulnerable to a CSRF attack that allowed illicit money transfers.
  - Popular video website **YouTube** was also vulnerable to CSRF in 2008 and this allowed any attacker to perform nearly all actions of any user.
  - **McAfee** was also vulnerable to CSRF and it allowed attackers to change their company system.

CSRF - Simple Requests

- The only allowed methods are:
  - GET
  - HEAD
  - POST
- What about the cookies?

GET/POST

- In HTTP GET the CSRF exploitation is trivial.
  - Why?
- HTTP POST has a different vulnerability to CSRF, depending on detailed usage scenarios:
  - In simplest form of POST with data encoded as a query string (field1=value1&field2=value2) CSRF attack is easily implemented using a simple HTML form and anti-CSRF measures must be applied.

CSRF-GET

http://bank.com/xfer?amount=500&to=attacker

CSRF-POST

SOP: the JavaScript code in one server can't fetch data from another server (Same Origin Policy)

Attack Example 2: CSRF with a form

On evil.com:

```
<form method="post" action="http://bank.com/transfer">
      <input type="hidden" name="to" value="ciro">
      <input type="hidden" name="amount" value="100">
      <input type="submit" value="Click to see cat photos">
</form>
```

If the user clicks the submit button, the browser will permit this. The SOP alone *does not forbid* this type of use.

It is the **SOP + synchronizer token pattern (STP)** that prevents that from working

Same-Origin Policy (link)

- Browsing multiple webpages poses a security risk.
  - For example, if you have a malicious website (www.evil.com) and Gmail (www.gmail.com) open, you don't want the malicious website to be able to access any sensitive emails or send malicious emails with your identity.

- Modern web browsers defend against these attacks by enforcing the *same-origin policy*, which isolates every webpage in your browser, except for when two webpages have the same origin.

Origins (link)

- The origin of a webpage is determined by its *protocol*, *domain name*, and *port*.
- For example, the following URL has protocol **http**, domain name **www.example.com**, and port **80**.
- Example:
  - http://wikipedia.org/a/ and http://wikipedia.org/b/ have the same origin. The protocol (http), domain (wikipedia.org), and port (none), all match. Note that the paths are not checked in the same-origin policy.

CSRF Vulnerable websites

- Web applications are at risk *that perform actions based on input from trusted and authenticated users without requiring the user to authorize the specific action*.

- A user who is *authenticated by a cookie saved in the user's web browser could unknowingly send an HTTP request to a site that trusts the user* and thereby causes an unwanted action.

CSRF Countermeasures

CSRF Countermeasures

- STP (Synchronizer Token Pattern)
- Identify Source Headers
  - Origin Header
  - Referer Header

CSRF Defenses - STP

Synchronization (Secret) Token Validation

- STP is a technique where a *token, secret and unique value for each request*, is embedded by the web application in all HTML forms and verified on the server side.

- The token may be generated by any method that ensures *unpredictability and uniqueness* (e.g. using a *hash chain of random seed*).
  - The attacker is thus unable to place a correct token in their requests to authenticate them.

CSRF Defenses – STP (cont.)

Example of STP set by Django in a HTML form:

```
<input type="hidden" name="csrfmiddlewaretoken" value="KbyUmhTLMpYj7CD2di7JKP1P3qmLlkPt" />
```

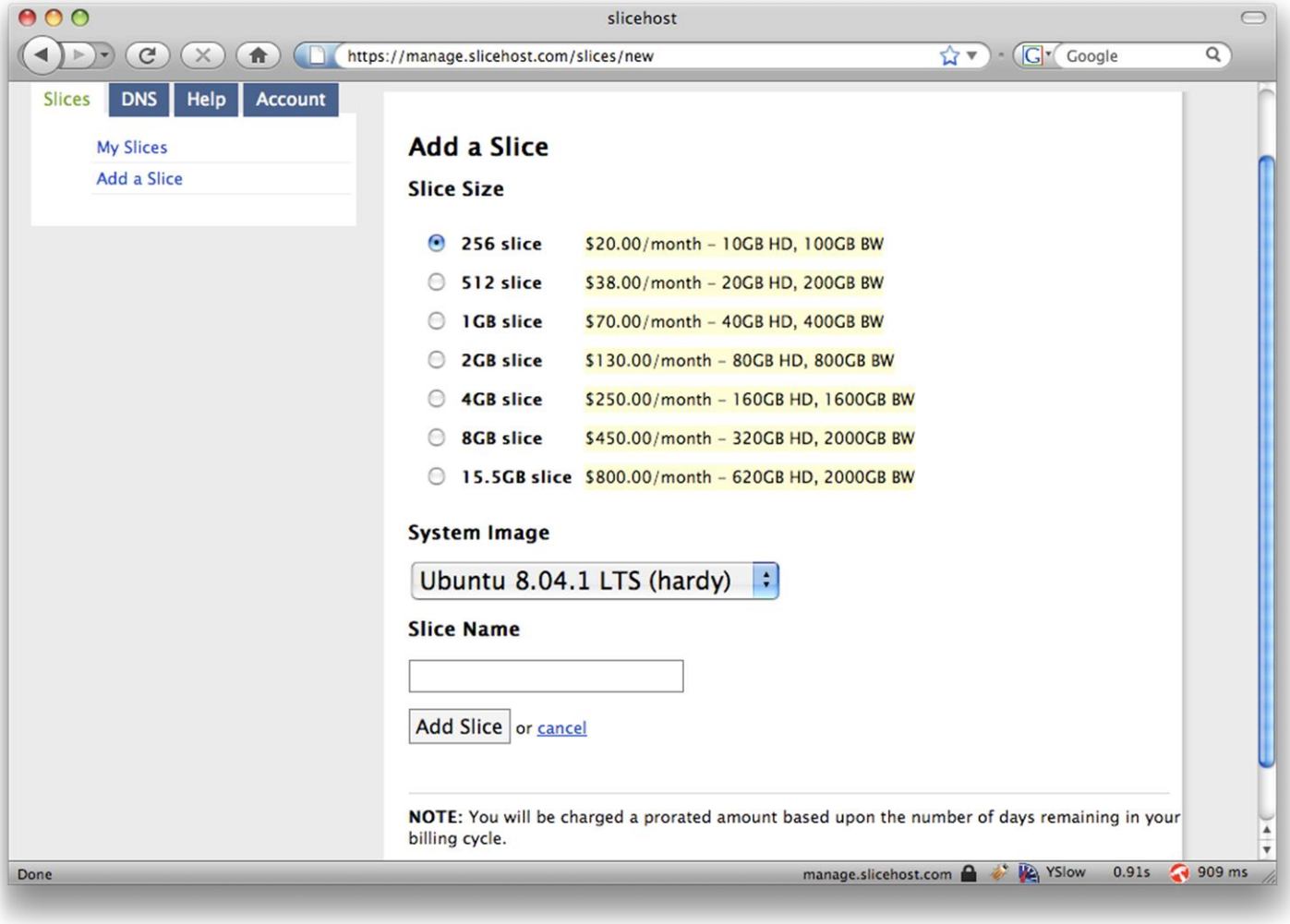
For every form on `bank.com`, generate a one time random sequence as a hidden parameter, and only accept the request if the server gets the parameter.

E.g., Rails' HTML helpers automatically add an `authenticity_token` parameter to the HTML, so the legitimate form would look like:

```
<form action="http://bank.com/transfer" method="post">
  <p><input type="hidden" name="authenticity_token" value="j/DcoJ2VZvr7vdf8CHKsvjdlDbmiizaOb5E
  <p><input type="hidden" name="to"                 value="ciro"></p>
  <p><input type="hidden" name="ammount"            value="100"></p>
  <p><button type="submit">Send 100$ to Ciro.</button></p>
</form>
```

So if `evil.com` makes a post single request, he would never guess that token, and the server would reject the transaction.

CSRF Defenses – STP




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

CSRF Defenses – STP

- Variations
  - Session identifier
  - Session-independent token
  - Session-dependent token
  - HMAC of session identifier

CSRF Defenses – Identifying Source Headers

- To identify the source origin, OWASP recommends using one of these two standard headers that almost all requests include one or both of:
  - Origin Header
  - Referer Header

CSRF Defenses – Origin Header

- If the Origin header is present, verify its value matches the target origin.
  - The Origin header in a HTTP request indicates where the request originated from
  - The Origin HTTP Header standard was introduced as a method of defending against CSRF and other Cross-Domain attacks.
  - It contains only origin properties of URL (avoid leaking URL parameters to other hosts)
    - ***Scheme*** (http or https)
    - ***Hostname*** (IP address or domain)
    - ***Port*** (i.e. 80 for http)

CSRF Defenses – Referer Header

- If the Origin header is not present, verify the hostname in the Referer header matches the target origin.
- If you click on a link, the URL of the current page is sent in the Referer header to the requested link.
- Checking the Referer is a commonly used method of preventing CSRF on embedded network devices because it does not require any per-user state that was in case of STP.
- Referer: a useful method of CSRF prevention when memory is scarce or server-side state doesn't exist.
- Referer header leaks the whole URL to other domains. If the URL contains sensitive data such as the session token or some other identifier, that is leaked when the URL is sent in the Referer header when the user clicks a link.

CSRF Defenses – Referer Header

Its typical use is thus: if I click on a link on a website, the referer header tells the landing page which source page I came from.

```
Source URL = www.mysite.com/page1 -> Target URL = www.example.com
referer = "www.mysite.com/page1"
```

It's heavily used in marketing to analyse where visitors to a website came from, and also very useful for gathering data and statistics about reading habits and web traffic.

However, it presents a potential security risk if too much information is passed on.

In the referer header's original RFC2616, the specification lays out that: "Clients SHOULD NOT include a Referer header field in a (non-secure) HTTP request if the referring page was transferred with a secure protocol" That is, if our request goes from https to http, the referer header should not be present.

However, RFCs are not mandatory, and data can be leaked. Facebook fell foul of this a little while ago, when it turned out that in some cases the userid of the originating page was being passed in the referer header to advertisers when a user clicked on an advert.

- Referer may leak privacy-sensitive information

  `http://intranet.corp.apple.com/projects/iphone/competitors.html`

Appendix – Helpful Links

- Understanding the iframe within HTML:
  - HTML Iframes
- Understanding Cookies:
  - Cookies and Session Management | Computer Security
- CSRF Prevention Cheat Sheet:
  - Cross-Site Request Forgery Prevention - OWASP Cheat Sheet Series
- Web Application Security, Testing & Scanning
  - Web Application Security, Testing, & Scanning - PortSwigger