Lab Practice

Input Validation

- Download Lab Week 5 Instruction file from course webpage on GitHub or Edmodo.

https://hogeschool.github.io/INFANL01-9/

Read the Instructions and perform the tasks.

10 minutes

10:00

Stop

Web Application Security

Lesson 5:

Web Trojans

What is a Trojan?

Trojan horse is something that appears to be a gift, but that actually is a trap.

In the context of computer security, the term has materialized into "a program that appears to be cool, but perform some scary damages, e.g. erases all files."

The "Trojan" part comes from the Greek legend of the Trojan horse.

What is a Computer Trojan?

Example 1

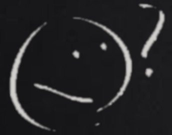
Imagine a voting web site allowing users to choose among different alternatives to collect statistics.

```
<form action="http://www.voting.example/vote.asp"
      method="get">
  <input type="radio" name="alt" value="1"/>Foo<br/>
  <input type="radio" name="alt" value="2"/>Bar<br/>
    :
</form>
```

Since the form uses the GET method, users voting will visit URLs like this when they submit the form:

```
http://www.voting.example/vote.asp?alt=2
```

How it can be attacked?



An attacker can easily copy the URL and mail it to lots of people and mislead them into the link.

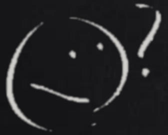
If several of the targeted people follow the link, they all give a vote after the attacker's liking, and the statistics get all wrong.

Example 2

The really scary threat with Web Trojans is that they work with authentication too.

As these trojans function by tricking a user or his browser into visiting a site, things will be done on behalf of the user if he is already logged in to the target site.

What happens if the user who is logged in to his on-line bank, visit this HTML in the browser:



```
<form name="f" action="https://www.bank.example/pay.asp"
      method="post">
  <input type="hidden" name="from-account"
                       value="1234.56.78901"/>
  <input type="hidden" name="to-account"
                       value="9876.54.32109"/>
  <input type="hidden" name="amount"
                       value="10000.00"/>
</form>
<script>document.f.submit()</script>
```

AUTOMATIC AUTHENTICATION

So, if the victim is already logged in to the bank, and the bank accepts that kind of form, the user will unintentionally transfer some money to the attacker, or to someone the attacker wants to get trouble into with the authorities.

For this to work, one will have to reach the victim while he is logged in to the target site.

- If the user is not always logged in, the attacker will somehow have to make sure the victim is signed on to the target site before tricking him into viewing the malicious HTML.
- How?

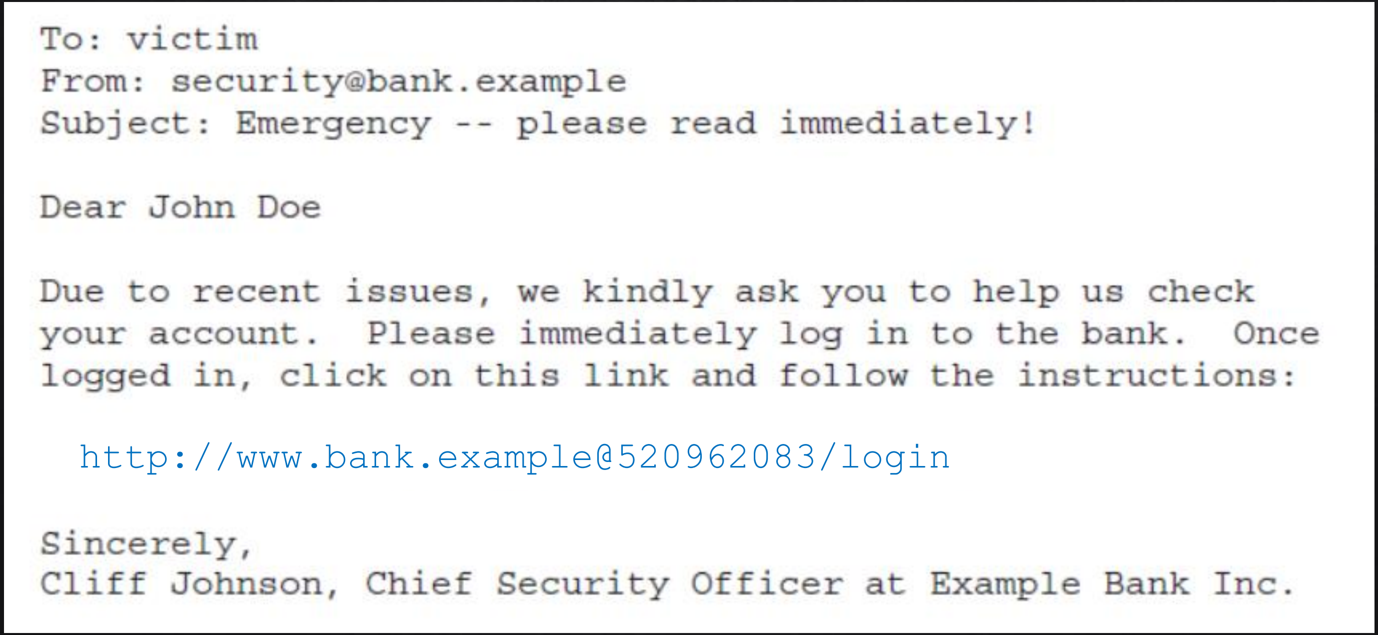
USER SHOULD BE LOGGED IN

- Sometimes that is quite easy:
  - if the user has clicked a "remember me" option, he will always be logged in.
  - if the target site is an intranet solution based on domain authentication.
  - if single sign-on solutions (like Google, Microsoft Passport, etc.) users may be logged in to more sites than they imagine.
- For a site where the attacker may add content, e.g. a discussion forum, that may be easy:
  - just add a very tempting note in the forum asking people to take a look at a link. Anyone reading the note is guaranteed to be logged in:
- Social Engineering to the rescue!

EXAMPLE

The target site is our bank named www.bank.example

The attacker can send the victim an E-mail that appears to come from the bank:



To: victim
From: security@bank.example
Subject: Emergency -- please read immediately!

Dear John Doe

Due to recent issues, we kindly ask you to help us check your account. Please immediately log in to the bank. Once logged in, click on this link and follow the instructions:

http://www.bank.example@520962083/login

Sincerely,
Cliff Johnson, Chief Security Officer at Example Bank Inc.

Problem

The Problem

Many web sites, including banks, shops, discussion sites, and whatnot are vulnerable to some kind of Web Trojan trickery.

- To see how to design a web solution that is not vulnerable, we need to understand the problem.

When someone browses our site, we typically generate web pages that contain URLs and forms inviting the user to do something.

- Web Trojans work because it is possible for attackers to give victims these offers on our behalf.

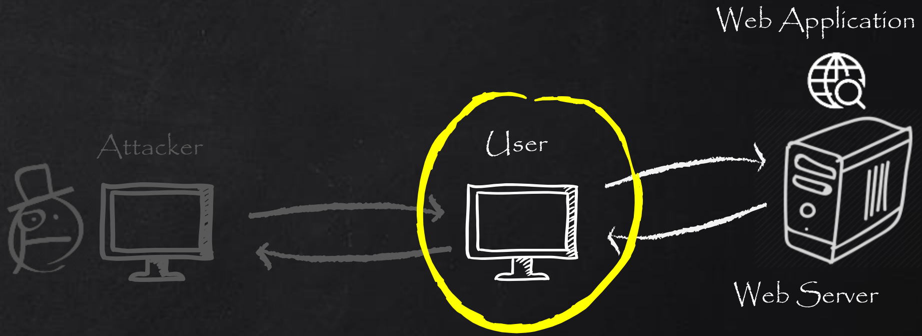
To avoid the threat, we need to make sure the action a user takes really is based on an offer we once gave him, rather than on an offer given him by someone else.

- How?

USE REFERER?

Many developers think that the Referer header is a good thing to check to make sure the visitor came from our site.

- In general, the Referer header should not be used for security, as it comes from the client-side.



But in the Web Trojan case it could have been useful, if it was not for the fact that many filter it out for privacy reasons.

- Referer headers are thus often missing in totally legitimate requests, so we need to find a method that does not depend on it.

REAUTHENTICATION?

An approach that would work would be to require reauthentication of the user for every action that changes something.

This solution includes adding a password field to every form presented to the client—an approach taken by many online banks.

Unfortunately, giving passwords all the time is cumbersome, so we will try a different approach.

Solution

A Practical Solution: ticket system

To protect against Web Trojans, developers should implement a "ticket system".

Central to this system are nonpredictable, random numbers, called tickets.

A web page may typically contain one or more offers to do an action that has side effects.

For each such offer, generate a unique, random string, and connect it with the offer.

If the offer is a form:

```
<input type="hidden"
       name="ticket" value="uFnVB5oHiMVMcFTN"/>
```

If the offer is a link:

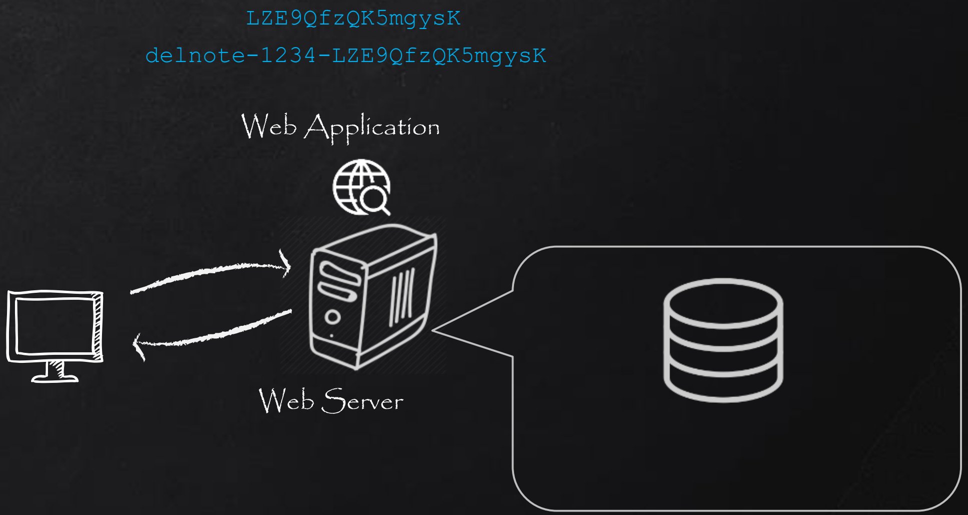
```
<a href="vote.jsp?alt=1&ticket=bVGZTa78LV00Zn9n">
   Yes, I agree</a>
```

GENERATE THE TICKET

For each ticket generated, add a string naming the action it refers to, and store the combined string in a ticket pool in the session of the user who receives the offer.

If the action in question is, for instance, to confirm deletion of a note numbered 1234, and the ticket is represented as LZE9QfzQK5mgysK, the string to store could be delnote-1234-LZE9QfzQK5mgysK.

You now have the same ticket on both the client-side and the server side.



Perform the Relevant Action

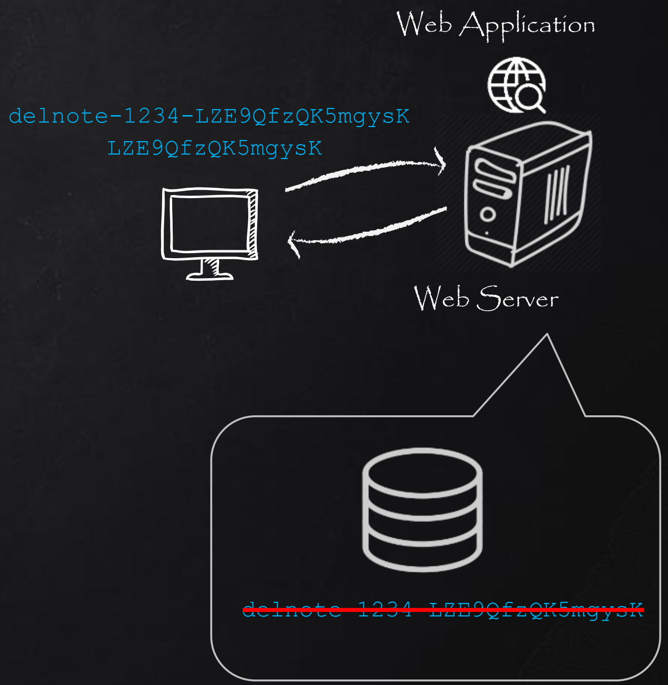
Whenever a request to perform an action arrives from the user, extract the ticket from the request.

`delnote-1234-LZE9QfzQK5mgysK`

Then add the name of the action that is about to be performed to the beginning of the string, and look for a match in the session ticket pool.

If a matching ticket is found, you may assume that the offer was given by your web site.

- In that case, perform the action, and remove the ticket from the pool.


Web Application
delnote-1234-LZE9QfzQK5mgysK
LZE9QfzQK5mgysK
Web Server
~~delnote-1234-LZE9QfzQK5mgysK~~

GOTCHAS

The ticket system works because attackers will not be able to guess what ticket values you may have given to the user, and they will not be able to insert tickets into the victim's ticket pool on the server side.

1. If you include the tickets in GET requests, so that they become part of URLs, you risk ticket leakage through Referer headers if the user follows a link from your site to other web servers.
2. The system will break if your application is vulnerable to Cross-site Scripting. If an attacker is able to insert JavaScript in a page generated by your server, he will be able to extract tickets from the page.

Hints in Ticket System Implementation

1. Web pages with tickets in them cannot be cacheable, as each ticket can only be used once.
   - Tell browsers and proxies not to cache pages with tickets in them to make sure every page comes with fresh, valid tickets.
2. Put an upper limit on the number of tickets in a pool.
   - There will be left-over tickets, as users do not necessarily follow up on all the offers we give them.
   - When the limit is reached, remove the oldest ticket and add the new one.
   - The limit will stop people from deliberately filling up memory with unused tickets.

3. Session timeouts make the server-side ticket pools disappear.
   The result is that we may get legitimate incoming requests with no matching ticket on the server, for instance from a user who has spent an hour filling in lots of details in a web form.
   Users won't be happy if we throw away their input.
   Instead we could redisplay the web page with a new ticket and all the incoming text filled in, tagged with an explanatory message that asks him to confirm that he really intends to perform the action.

4. Tickets are needed only for requests that actually change something on the server.

   A user who wants to edit a note, for example, will first request the note editing form, typically by clicking a link.

   This request does not change anything, so it need not be protected by a ticket.

   When he has finished editing, he POSTs his changes.

   The second request updates the server-side database, so it should be protected by a ticket.

Summary

Summary

- Attackers may give their victims offers on behalf of a target web site and thereby trick them into doing something they never intended to do.
- The offers may be URLs or auto-submitting forms, and they may be given through any channel available, such as E-mails and web pages outside of the target site.
- To protect a web site and its users from these Web Trojans, web developers will need to implement special security mechanisms, for instance the ticket system.

Summary

- The ticket system will make sure an action taken is based on an offer given by the web site rather than by some off-site attacker.
- As of writing, mechanisms to protect against Web Trojans are not commonly in use, making lots of web sites vulnerable.

Your tasks for this week

Reading:

- "Innocent Code: A Security Wake-Up Call for Web Programmers (Chapter 5).

Quiz:

- WAS week 5 on the course page on Edmodo.

Note that Quizzes are accessible every Thursday to Sunday only.