Hacking Techniques

Class demo using “Hacme Bank”
Dr. Joseph Gersch

With thanks to Foundstone for creation of this tool for teaching hacking
Techniques Demonstrated

- SQL Injection
- Database enumeration
- Create a new account by using SQL Injection
- Horizontal Privilege Escalation
- Vertical Privilege Escalation
- XSS (Cross Site Scripting)
- Using a proxy (man-in-the-middle data tampering) to steal $1,000,000
Bypass the LOGON with simple SQL Injection

- Login field: ' OR 1=1 --
- Effect was to log me in without a password as the first user in user table
Used SQL Injection: ‘HAVING 1=1 --

Error comments sent to web browser revealed the table name and one of the columns:

“column fsb-users.user_id is invalid…”

Brute force, good guesses, and other injection techniques can help reveal other column names such as user_name, login_id, password, creation_date
Create a new user & account

- Don’t want to use a real bank user’s account; the FBI will find you

- Use SQL Injection: `SELECT INSERT INTO FSB_USERS(user_name, login_id, password, creation_date) VALUES('hacker', h12345', 'mypw', GETDATE()); --`

- Can now log in as this new user

- A similar technique can create a user account for money transfers
Horizontal Privilege Escalation  
(access another user’s records)

- The URL for viewing my accounts used *query variables*:  
  - …?function=TransactionDetails&account=5204320422040004

- If I changed the URL query variable account to  
  5204320422040008 (or other account numbers not owned by me), the web page displayed information for other users.

- There was no sanity checking of the URL
Vertical Privilege Escalation
(becoming administrator)

- Again, change the URL query variable
  - Login
  - Query variable says “…?function=Welcome”
  - Change to “…?function=admin\Sql_Query”
  - And now I have administrator access to the database and can create accounts, users, and lots of other damage

- This can only be done after much experimentation to determine this query variable even exists
The bank site has a mechanism to post messages that other users can view. These messages are persistently stored in the database.

Instead of posting a text message into the form such as “I like this bank”, post javascript: `<script> alert(“Houston we have a problem”) </script>`. All users viewing messages will get the alert because it is stored in the database.

To steal the cookie: `<script> alert(document.cookie) </script>`

This will reveal session info. Normally alert would not be used, but instead transmit the cookie to a remote server or public bulletin board so it would be invisible.
We attempted to transfer a negative dollar amount from hacker’s account to legitimate account. This failed because the web browser had javascript that disallowed a negative amount.

Since the Browser did *SANITY CHECKING*, maybe the designers forgot to do sanity checking at the application server itself.

So we used a proxy (man-in-the-middle) to tamper the data after submitting a form with a positive dollar amount into a negative dollar amount. This was then passed on to the web server. The application gladly accepted the negative amount and transferred $1,000,000 to the hacker account.

Transfer the money to the Cayman Islands, delete all accounts and get out!!!!
Caveats

- Don’t do this to a real bank. Remember my warnings!
- Most modern financial Web Applications have put in sanity checks and other security checks to prevent these types of attacks.
- But many web sites have not put in security checks and can be hacked.
- Again… don’t do this on the real Internet. You will get into trouble. Serious trouble.