A Case Study of
Django

Web Applications that are Secure by Default

Mohammed ALDOUB

@Voulnet

Web Security Essentials

- The essentials of web application security are still not well understood.
- Most developers have little to no idea about web security fundamentals.
- Higher adoption to new web technologies, but no accompanying security awareness.

Web Security Essentials

- The basic idea of web security: **Never** trust users, and never trust their data. **No exceptions**.
- Many **layers** exist in web technologies, and therefore many attack vectors and possibilities.
- Web developers must **understand** risks and mitigations for **all** web layers.

Problems in Applying Web Security

- Web security cannot be achieved if developers are not well trained in security. **Education** is key.
- **Deadlines** will almost always result in security vulnerabilities. Developers who are too busy and under pressure will not focus on security.
- Security is not integrated **early** in the development process, so it gets **delayed/forgotten**.

Bad Practices in Web Security

- Developers **don't validate** user input.
- Even if they validate it, they do it **poorly** or out of **context**.
- Developers make **wrong** assumptions about security:
  - "It's ok, we use SSL!"
  - "The Firewall will protect us"
  - "Who will think of attacking this function?"
- Most developers **copy/paste** code from the internet

Bad Practices in Web Security

- Session/password management is done poorly:
  - Sessions are easy to **forge** by attackers.
  - Passwords are stored as **plaintext**.

- **Server & Database** configuration/security are not understood by web developers.

- Developers don’t realize the threats on end users:
  - Cross Site Scripting (**XSS**)
  - Cross Site Request Forgery (**CSRF**)

django

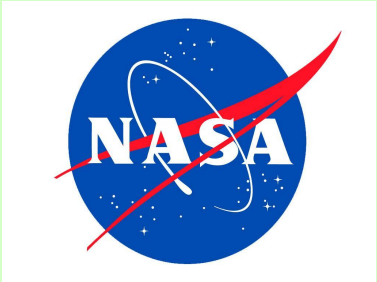
- Django is a Web Application Framework, written in Python
- Allows rapid, secure and agile web development.
- Write better web applications in less time & effort.

- Django is loaded with **security** features that are used by **default**.
- Security by default provides great **protection** to developers with **no security** experience
- Django makes it more **difficult** to write **insecure** code.

django

- Django is used by many popular

es/s



Security Features of Django

- Django provides many default protections against web threats, mainly against problems of:
  - User Management
  - Authorization
  - Cookies
  - SQL Injection
  - Cross Site Scripting (XSS)
  - Cross Site Request Forgery (CSRF)
  - Clickjacking
  - Files
  - E-mail Header Injection
  - Cryptography
  - Directory Traversal

User Management

- Developers make many mistakes in user management.
- Passwords are stored/transferred as plaintext.
- Users are exposed if databases get leaked.
- Weak authentication methods are used by inexperienced developers.

User Management

- Django provides a default User model that can be used in any website. It comes equipped with correct session management, permissions, registration and login.
- Developers don’t need to re-invent the wheel and re-introduce user management risks.
- Django provides strong password hashing methods (bcrypt, PBKDF2, SHA1), with increasing work factors.
-

User Management

- Django provides easy methods for user management such as is_authenticated(), permission_required(), requires_login(), and more, offsetting difficult session and permission code away from the developer.

- Django provides secure default password reset and login redirection functionality. Developers don’t need to create password reset forms and introduce risks.

- By using Django’s user management module, developers will not make mistakes such as

Clickjacking

- Clickjacking is an attack where an attackers loads an invisible page over a visible one. The user thinks he is clicking on the visible page, but he’s actually clicking on invisible buttons and links.

- Can be used to trick users into buying items, deleting content or adding fake friends online.

- Django provides direct protection against Clickjacking attacks using the X-Frame-Options header. Only one line of code!

Clickjacking Example

Image taken from ‘Busting Frame Busting’ research paper (found in references)

Cross Site Scripting (XSS)

- XSS is one of the most dangerous and popular attacks, where users instead of servers are targeted.
- In an XSS attack, an attacker runs evil scripts on the user’s browser, through a vulnerable website.
- It can be used to steal cookies, accounts, install malware, deface websites and many more uses.

Cross Site Scripting (XSS)

- XSS is very easy to introduce by ignorant developers, example:

```
<?php
echo "Results for: " . $_GET["query"];
?>
```

- It's okay if the search query was Car, but what if the attacker entered…

  <script>alert(document.cookie)</script>

Cross Site Scripting (XSS)

- Evidently XSS is a critical attack, so Django provides great default protections against it.
- HTML output is always escaped by Django to ensure that user input cannot execute code.
- Django’s templating engine provides autoescaping.
- HTML Attributes must always be quoted so that Django’s protections can be activated.

SQL Injection (SQLi)

- SQL Injection is a dangerous attack in which evil data is sent to the database to be executed as destructive commands.

- Developers write SQL queries in a wrong way, allowing attackers to inject SQL commands into the query, to be

```
string sql = “SELECT * FROM USERS WHERE name=‘” +
Request[‘username’] + “’”;
```

- Looks innocent, but what if the user entered ‘; DROP TABLE USERS;-- ?

SQL Injection (SQLi)

- SQL injection attacks are used to read and corrupt databases, take complete control over servers as well as modify web pages (and therefore steal user sessions or install malware)

- The good news is that Django provides excellent defense against SQL Injection!

- Django uses ORM and query sets to make sure all input is escaped & validated.
- Developers do not need to write any SQL. Just write Python classes and Django will convert them to SQL securely!

SQL Injection (SQLi)

- No matter where input comes from (GET,POST,COOKIES), Django will escape all input that goes to the database.

- Even if developers needed to write raw SQL, they can use placeholders like "Select * from users where id = %s ” which will safely validate input.

Cookies

- Django sets cookies to HttpOnly by default, to prevent sessions from getting stolen in most browsers.
- Session ID are never used in URLs even if cookies are disabled.
- Django will always give a new session ID if a user tried a non-existent one, to protect against session fixation.
- Cookies can be digitally signed and time-stamped, to protect against tampering of data

Files

- Django provides excellent protection to files.
- No webroot concept in Django. Only the directories and files you allow are requested. Even if attackers upload a file, it is only downloaded if you allow it in URLConf.
- Django executes Python code from the outside of the web root, so attackers cannot retrieve any files not explicitly allowed from the web root.

Cross Site Request Forgery (CSRF)

- CSRF is an attack where an attacker can force users of a website to perform actions without their permission.

- If a user is logged into website A, an attacker can let a user visit website B, which will perform actions on website A on behalf of the user.

- This happens because the forms in website A are not protected against CSRF.

- Basically CSRF means evil websites can let users of other websites perform actions without user permission.

Cross Site Request Forgery (CSRF)

- Example: A form in website A allows a logged in user to delete his account. If there is no CSRF protection, website B can force visitors to delete their account on website A.

- Example: Suppose website B has this HTML form in its code. What happens if a user of website A visits B?

```
<form
action="http://websiteA.com/deleteMyAccount.php”
method=”post” >
</form>
```

Cross Site Request Forgery (CSRF)

- The effects of CSRF is that attackers can make users perform ANY action on the vulnerable website.
- Django provides CSRF protections for all POST,PUT,DELETE requests (according to RFC2616).
- If website A used Django CSRF protection, the form would be:

```
<form action="/deleteMyAccount.php"
method="post" >
<input type='hidden' name='csrfmiddlewaretoken'
value='Aes4YiAfBQwCS8d4T1ngDAa6jJQiYDFs' />
</form>
```

E-mail Header Injection

- E-mail Header injection is a less popular attack that targets weak email-sending forms in websites.
- By crafting a special string, attackers can use your email form to spend spam through your mail server, resulting in your domains/IPs getting blocked and possible worse effects.

- Example email form:

To: mycustomer@example.com
Subject: Customer feedback
<email content here>

E-mail Header Injection

- What if the attacker supplies the following data as the email content? They will be able to use your website as a spam base.
- “\ncc: spamVictim@example.com\n<spam content>”
- It would be:

To: mycustomer@example.com
Subject: Customer feedback
cc: spamVictim@example.com
<spam message content, buy drugs, lose weight or something>

- Django provides default protection by using the built-in

Final Remarks

- It must be understood that nothing can protect developers if they refuse to learn about web security and vulnerabilities.
- The point of Django's default security features is to make it very easy to add security, and very difficult to remove security.
- However, developers still need to learn the basics of security and risk assessment.
- Knowledge is the best defense against web attacks.

References

- http://davidbliss.com/sites-built-using-django
- https://docs.djangoproject.com/en/1.4/topics/security/
- http://www.djangobook.com/en/2.0/chapter20/
- http://seclab.stanford.edu/websec/framebusting/framebust.pdf

Questions?

- Do not hesitate to ask any question!

- Do not hesitate to let your developers try Django in the workplace! It could be your road to increased productivity and security!