מכון טכנולוגי חולון
Holon Institute of Technology

"Communication LTD"

STRIDE & DREAD

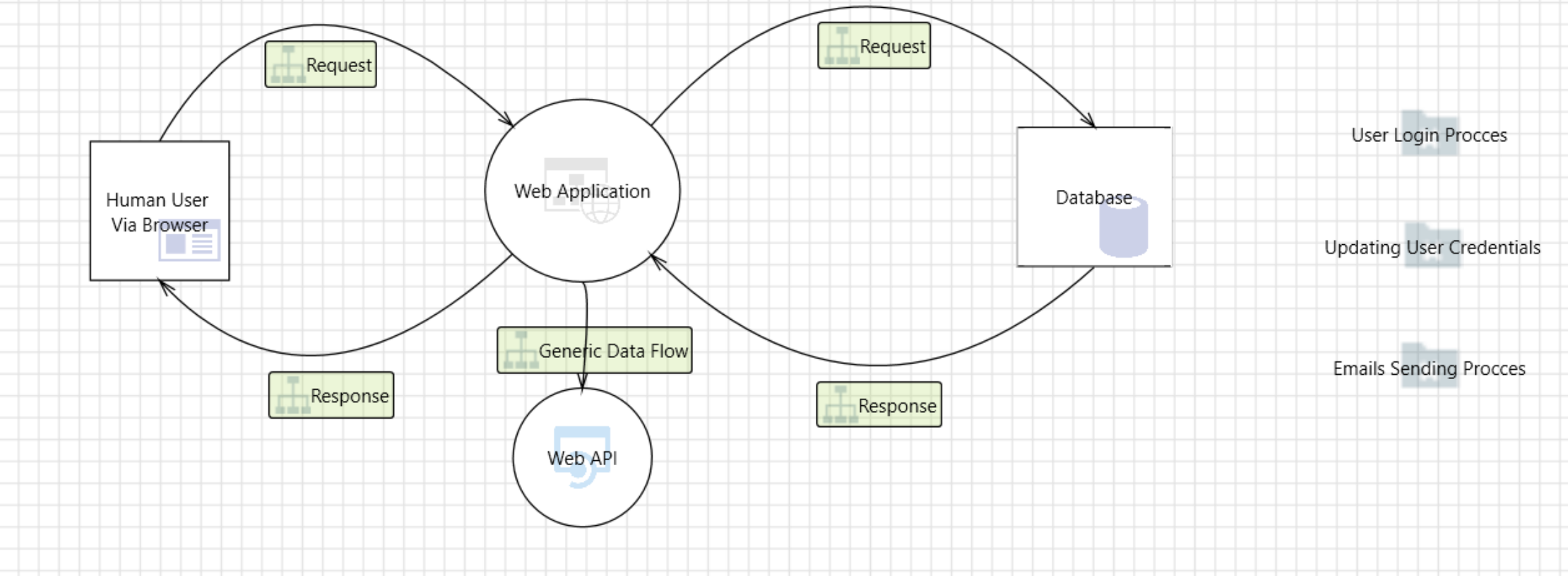
Threat Analysis Model

Computer Security Course
65337-01 | Roi Zimon

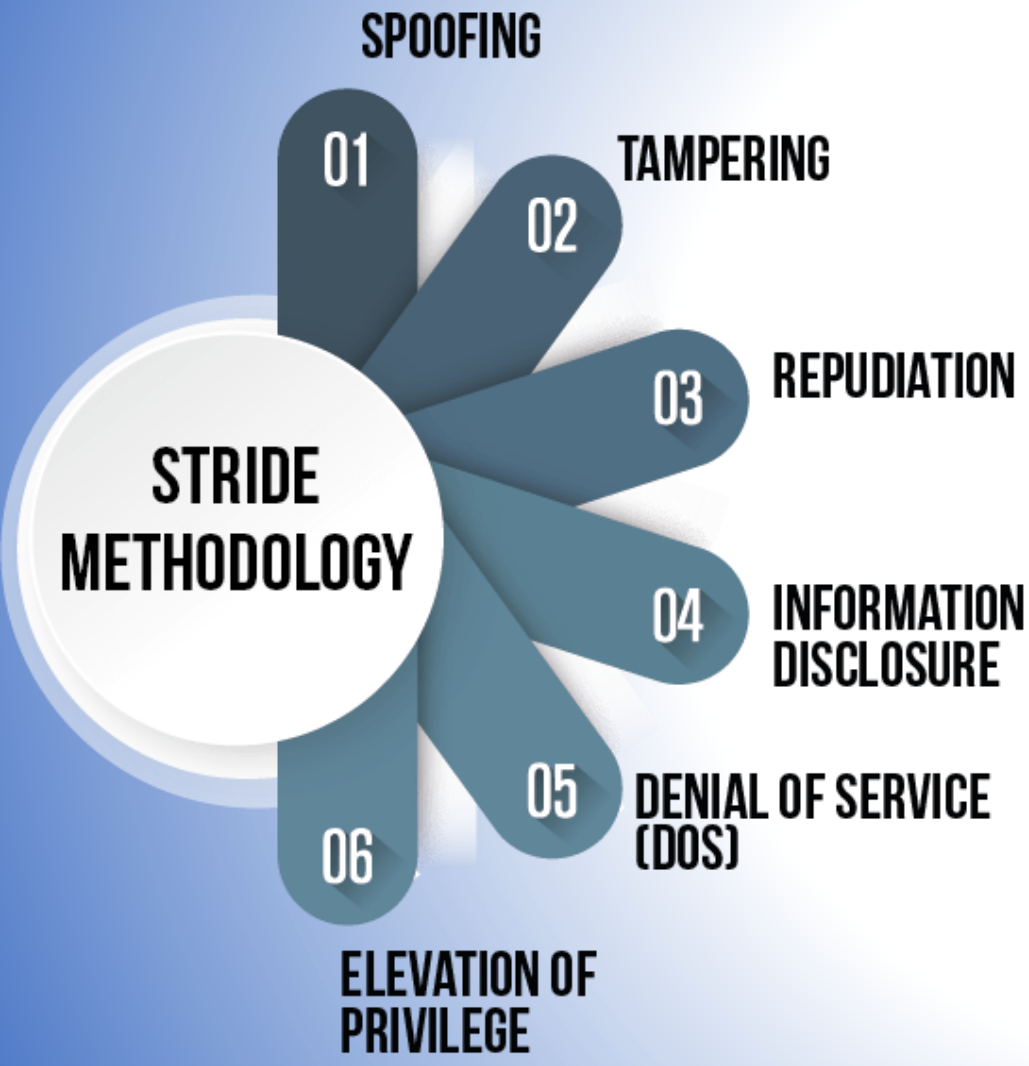
Students

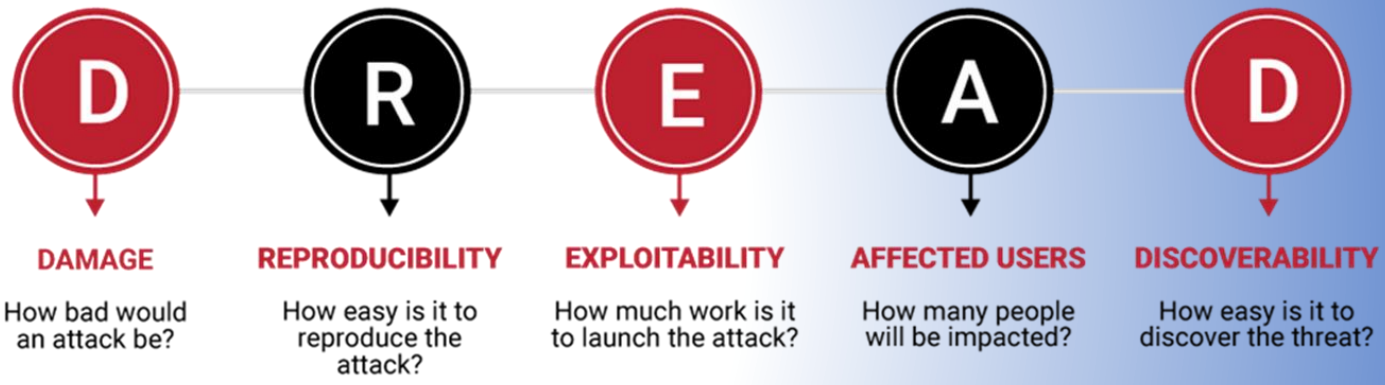
Name	ID
Roy Edri	315395988
Or Attias	207953308
Orian Dabod	308337062
Rotem Gershenzon	207495417
Nisan Cohen Burayev	315433656

Data Flow Diagram



STRIDE & DREAD





STRIDE MODEL

Application Threat Modeling using DREAD and STRIDE is an approach for analyzing the security of an application. It is a structured approach that enables you to identify, classify, rate, compare and prioritize the security risks associated with an application.

STRIDE stands for:

Spoofing

Tampering

Repudiation

Information Disclosure

Denial of Service

Elevation of Privilege

DREAD MODEL

The problem with a simplistic rating system is that team members usually will not agree on ratings. To help solve this, add new dimensions that help determine what the impact of a security threat really means.

The *DREAD* model is used to help calculate risks ratings for a given threat by asking the following questions:

Damage potential: How great is the damage if the vulnerability is exploited?

Reproducibility: How easy is it to reproduce the attack?

Exploitability: How easy is it to launch an attack?

Affected users: As a rough percentage, how many users are affected?

Discoverability: How easy is it to find the vulnerability?

Spoofing Threats

	Threat	Threat Analysis\ Solution
1	An adversary can create a fake website and launch phishing attacks	מומש, ע"י OPEN SSL על מנת ליצור סרטיפיקציות
2	An adversary may spoof Web Application and gain access to Web API	יש לנו MFA או TFA למשתמש הייעודי בגוגל לשירות המיילים, קשה להתחזות, בנוסף, יש בגוגל בקרה על מכשירים מחוברים
3	An adversary can create a fake website and launch phishing attacks	באופן עקרוני, ניתן להתחזות לאתר שלנו COMMUNICATION LTD
4	An adversary may spoof Database and gain access to Web Application	מימוש של SALT גם אם יודע אלג' הצפנה עדיין נדרש גם עם SALT בוצע HASH עם SALT

Tampering Threats

	Threat	Threat Analysis \ Solution
1	An adversary can deface the target web application by injecting malicious code or uploading dangerous files	מניעת XSS ב-Django
2	An attacker steals messages off the network and replays them in order to steal a user's session	יכולת להסניף בקשות, ישנם כלים זדוניים להסנפת התעבורה
3	An adversary can gain access to sensitive data by performing SQL injection through Web App	מפורט במצגת על התקפת SQL
4	An adversary can gain access to sensitive data stored in Web App's config files	קבצי קונפיגורציה לא נמצאים בגיטאהב (מופיעים ברידמי לצורכי למידה) - לא מוצפנים בשרת

Repudiation Threats

	Threat	Threat Analysis / Solution
1	An adversary can deny actions on database due to lack of auditing	לוגי MYSQL ניתן לראות אם מישהו תקף
2	Attacker can deny the malicious act and remove the attack footprints leading to repudiation issues	לוגים נשמרים בשרת, תוקף צריך הרשאות לשרת כדי לערוך את היסטוריית הלוגים ולמחוק
3	Attacker can deny the malicious act and remove the attack footprints leading to repudiation issues	לוגים נשמרים בשרת, תוקף צריך הרשאות לשרת כדי לערוך את היסטוריית הלוגים ולמחוק

Information Disclosure

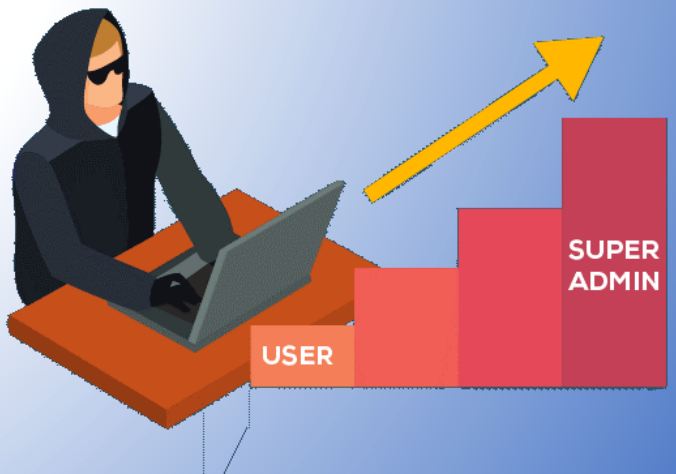
	Threat	Threat Analysis
1	An adversary can gain access to sensitive data by performing SQL injection	מפורט במצגת על SQLהתקפת בגאנגו
2	An adversary may gain access to unmasked sensitive data such as credit card numbers	אין לנו מידע כזה / שירות - אין כרטיסי אשראי או מידע רגיש אודות המשתמשים, באתר אמיתי עם הוראות קבע ללקוחות של חברה רוב הסיכויים שכן יהיה אחסון בצורה כזו או אחרת ויש לוודא שימוש במנגנון אבטחה ייעודי כגון: PCI DSS בעולם הפיננסי.
3	An adversary can reverse weakly encrypted or hashed content	לא לחשוף מימוש של הצפנות בהודעות שגיאה שמרמזות על אלגוריתם HASH או DB שבו השתמשנו

Denial of Service

	Threat	Threat Analysis
1	An adversary can perform action on behalf of other user due to lack of controls against cross domain requests	אין הגבלת קיבולת לכמות בקשות, לא בדקנו עמידה בעומסים של המערכת.

Elevation of Privilege

	Threat	Threat Analysis \ Solution
1	An adversary can gain unauthorized access to database due to lack of network access protection	מה נעשה ברמת ההגנה על DB - יש סיסמא לדטבייס, אין גישה לפורט 22 בקונטיינר, משמע לא יכול להתבצע שימוש ב SSH
2	An adversary can gain unauthorized access to database due to loose authorization rules	לא הוגדרו חוקים בדרישות הפרויקט
3	An adversary may bypass critical steps or perform actions on behalf of other users (victims) due to improper validation logic	יש בפרויקט עמוד ייעודי לאדמין רק ליוזרים שמוגדרים כאדמין.




Stored
XSS

Stored XSS is a vulnerability in web applications that allows the execution of illegitimate client-side scripts.

And from an attacker's perspective, an XSS attack is a technique where the attacker injects malicious client-side scripts into the web application. When the user requests the affected page, the malicious script is executed.

Malicious actors use XSS for various purposes, including these common occurrences:

- Stealing of sensitive information
- Identity theft
- Remote code execution

However, XSS attacks can originate from any untrusted source of data, such as cookies or Web services, whenever the data is not sufficiently sanitized before including in a page.

Django vs. XSS

- The developers of the Django framework have considered the security aspects.
- As a result, Django comes with built-in security features against XSS attacks.
- XSS attacks happen through injections—injection of scripts that contain HTML tags.
- We will explain in more details about those certain security features.

Django vs. XSS

Are Django applications vulnerable to XSS?

Using Django templates protects you against the majority of XSS attacks. However, it is important to understand what protections it provides and its limitations.

Django templates *escape specific characters* which are particularly dangerous to HTML. While this protects users from most malicious input, it is not entirely foolproof. For example, it will not protect the following:

<style class={{ var }}>...</style>

if var is set to 'class1 onmouseover=javascript:func()', this can result in unauthorized JavaScript execution, depending on how the browser renders imperfect HTML. (Quoting the attribute value would fix this case.)

In addition, if you are using the template system to output something other than HTML, there may be entirely separate characters and words which require escaping.

Django vs. XSS

Are Django applications vulnerable to XSS?

Basically, the application is using the input and adding it to the pre-decided text.
Once the application builds this result, it sends the response to the user's browser where the response is rendered. Any input might seem harmless. But let's see how this process could be dangerous. The first rule of web application security is never to trust user input.
If the input was <script>alert('XSS');</script>; ,
this injection wouldn't work in Django, which has an automatic *HTML escaping feature*. This feature converts certain HTML characters into their HTML code as follows:

- < is converted to <
- > is converted to >
- ' (single quote) is converted to '
- " (double quote) is converted to "
- & is converted to &

Django converts the input to <script>alert('XSS');</script>. And when this is rendered on the browser, it doesn't execute because it's not a script anymore.

Django XSS Examples

Unsafe Use of “Safe”

Django by default escapes data, but it provides a safe filter you can use to disable escaping for particular data. The safe filter is used mostly when data is known to be safe. If a malicious actor finds out that you’ve been tagging a data value as safe, they could inject their malicious script in that data field. And because it’s marked as safe, Django won’t escape it. As a result, when the data with the malicious script is rendered, an XSS attack is performed.

See implementation: (line 32-33)

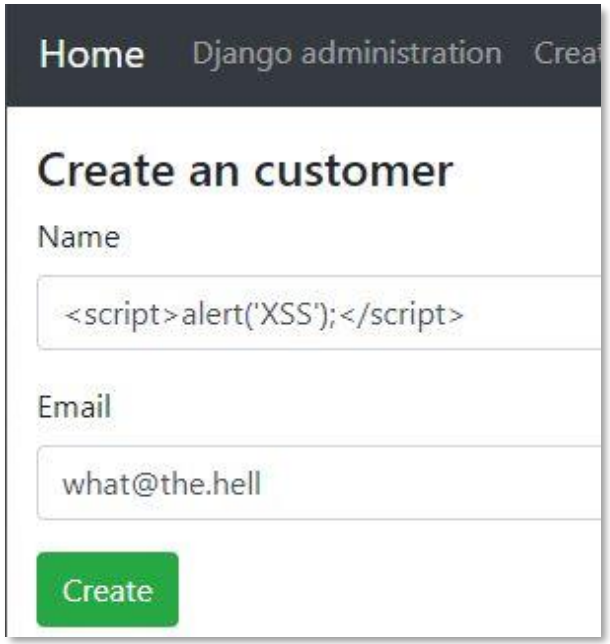
source/accounts/templates/accounts/customer_create.html

Django XSS (Project Example)

Unsecure Configuration:

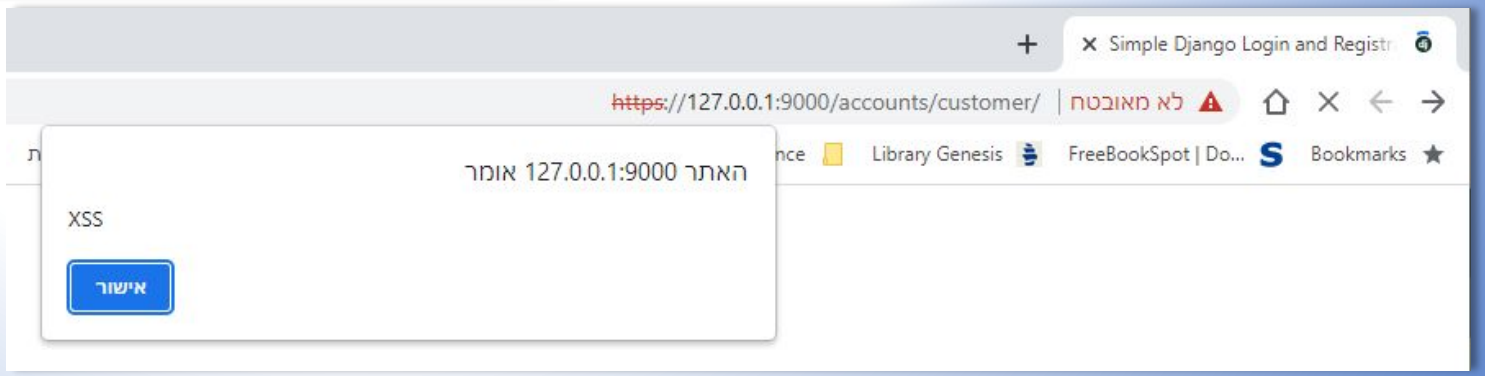
```
web:
  container_name: application
  build: ./
  command: uwsgi --master --https
  ports:
    - "9000:8000"
  environment:
    BWAPP_SQLI: False
    BWAPP_XSS: False
    CORS_ORIGIN_ALLOW_ALL: False
```

Result:



Customers Registered in Comunication_LTD

Customer Name	Customer Email
<script>alert('XSS'); </script>	what@the.hell
<script>alert('XSS'); </script>	what@the.hell



Django XSS Prevention

Safe Filter

Misuse of the safe filter can be dangerous, as depicted in the example above. Hence, it's very important to be smart and careful in using it.

Don't use this filter if it's not necessary.

If the application is developed, find all the instances of this filter.

You can then evaluate if it's really necessary. If it's necessary, then analyze its impact on security and implement additional measures.

See implementation: (line 32-33)
source/accounts/templates/accounts/customer_create.html

SQL Injection

SQL injection is a vulnerability in which malicious data is injected into the application and sent to a SQL database as part of a SQL query and the database executes the malicious query.

Django vs. SQL Injection

Are Django applications vulnerable to SQL Injection?

A Django application is by default protected against SQL Injection as it uses Object Relational Mapping (ORM).

ORM - a developer does not need to write direct SQL queries, but instead uses the in-built QuerySet APIs.

But in the end, it is still an SQL query. Can't an attacker inject a malicious input and send it as part of that query?

Django then converts the Python query to SQL query and communicates with the database.

Django's official documentation states that SQL queries are constructed using *query parameterization.*

A query's SQL code is defined separately from the query's parameters. Since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver.

Django vs. SQL Injection

NOTE !

Django also gives developers power to write raw queries or execute custom sql.
These capabilities should be used sparingly, and you should always be careful to properly escape any parameters that the user can control.

Django provides great protection against SQL Injection most of the times. The developers must however exercise caution and use the prevention mechanisms mentioned above while using raw(), extra(), RawSQL and direct SQL queries.

How to detect?

SQL injection can be detected in two ways:
Performing Source Code Security Review & Performing Web Application Vulnerability Scanning

Examples of Static Application Security Testing (SAST) checks.

Django vs. SQL Injection

Additional SAST Options (Not Implemented in Project)

Examples of Static Application Security Testing (SAST) checks.

Performing Source Code Security Review, it has the following tests for identifying the flaw:
B608 - hardcoded_sql_expressions
B610 - django_extra_used
B611 - django_rawsql_used

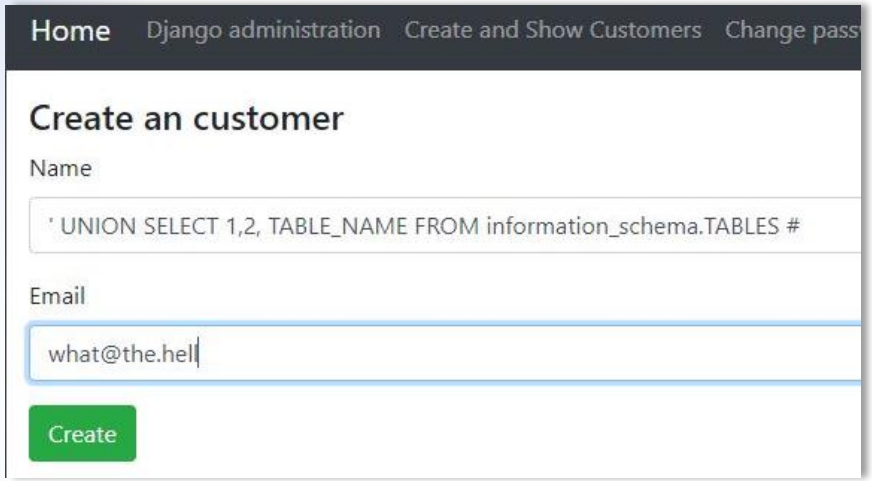
SSlabs – Option for DAST checks.

Django SQL Injection (Project Example)

Unsecure Configuration:

```
web:
  container_name: application
  build: ./
  command: uwsgi --master --https
  ports:
    - "9000:8000"
  environment:
    BWAPP_SQLI: False
    BWAPP_XSS: False
    CORS_ORIGIN_ALLOW_ALL: False
```

Result:



Customers Registered in Comunication_LTD

Customer Name	Customer Email
' UNION SELECT 1,2, TABLE_NAME FROM information_schema.TABLES #	what@the.hell
' UNION SELECT 1,2, TABLE_NAME FROM information_schema.TABLES #	working@allright.com

Thank You !