Offensive Web Security

CS 390 – Web Application Development

J. Setpal

November 29, 2023

Outline

Outline

WIWYT – Offensive Security

- Adversarial approaches enable a very effective way to build security. It does not rely on assumptions.
- It is extremely fun.

WIWYT – Offensive Security

- Adversarial approaches enable a very effective way to build security. It does not rely on assumptions.
- It is extremely fun. Also very valuable: bug-bounty programs award 5-figure payouts for critical vulnerability disclosures.

Outline

Symmetrical and Asymmetrical Encryption

Here's a sample encryption workflow:

Symmetrical and Asymmetrical Encryption

Here's a sample encryption workflow:

Q: Is this secure from intrusion?

Symmetrical and Asymmetrical Encryption

Here's a sample encryption workflow:

Q: Is this secure from intrusion?
A: Not really! Anyone on the same network can sniff keys, and security is thwarted.

Symmetrical and Asymmetrical Encryption

Here's a sample encryption workflow:

Q: Is this secure from intrusion?
A: Not really! Anyone on the same network can sniff keys, and security is thwarted.

We can improve this by making the keys *asymmetrical* – but this requires us to setup an accompanying security policy.

Public Key Encryption

We setup two keys:

a. Public Key

b. Private Key

Public Key Encryption

We setup two keys:

a. Public Key

b. Private Key

These are really the same! We just set one as private, and one as public.

Public Key Encryption

We setup two keys:

a. Public Key

b. Private Key

These are really the same! We just set one as private, and one as public.

The workflow is as follows:

a. We first send a request to the server, to ask for their public key.

Public Key Encryption

We setup two keys:

a. Public Key
b. Private Key

These are really the same! We just set one as private, and one as public.

The workflow is as follows:

a. We first send a request to the server, to ask for their public key.
b. We then encrypt our session symmetric key with the private key, and send it to the server.

Public Key Encryption

We setup two keys:

a. Public Key
b. Private Key

These are really the same! We just set one as private, and one as public.

The workflow is as follows:

a. We first send a request to the server, to ask for their public key.
b. We then encrypt our session symmetric key with the private key, and send it to the server.
c. The server decrypts the key, and can now read information without sending plaintext.

Public Key Encryption

We setup two keys:

a. Public Key

b. Private Key

These are really the same! We just set one as private, and one as public.

The workflow is as follows:

a. We first send a request to the server, to ask for their public key.

b. We then encrypt our session symmetric key with the private key, and send it to the server.

c. The server decrypts the key, and can now read information without sending plaintext.

Q: Is this secure from intrusion?

Public Key Encryption

We setup two keys:

a. Public Key
b. Private Key

These are really the same! We just set one as private, and one as public.

The workflow is as follows:

a. We first send a request to the server, to ask for their public key.
b. We then encrypt our session symmetric key with the private key, and send it to the server.
c. The server decrypts the key, and can now read information without sending plaintext.

Q: Is this secure from intrusion?
A: Still no! If an intruder can inject their *own* public key instead of the server, there is no way for the client to know. Security thwarted.

Certificate Authorities

A certificate authority is a trusted (eg. `https://letsencrypt.org/`) entity that stores and signs digital keys (certificates).

Certificate Authorities

A certificate authority is a trusted (eg. `https://letsencrypt.org/`) entity that stores and signs digital keys (certificates).

The certificate authority signs, or verifies the certificate of a given server.

Certificate Authorities

A certificate authority is a trusted (eg. `https://letsencrypt.org/`) entity that stores and signs digital keys (certificates).

The certificate authority signs, or verifies the certificate of a given server.

Q: But what about verifying the certificate authorities' key? Is that safe?

Certificate Authorities

A certificate authority is a trusted (eg. `https://letsencrypt.org/`) entity that stores and signs digital keys (certificates).

The certificate authority signs, or verifies the certificate of a given server.

Q: But what about verifying the certificate authorities' key? Is that safe?
A: We **recursively verify certificates**, until we leverage a pre-installed root certificate, that is self-signed by the local machine.

Outline

Offensive Security

So far, we have discussed mitigations to vulnerabilities.

Offensive Security

So far, we have discussed mitigations to vulnerabilities. Things like:

1. Using encrypted communication (HTTPS).
2. Password Salting & Hashing.
3. Using OAuth2.
4. Setting strict password policies.
5. Using stable node libraries.
6. Services to track and report vulnerabilities (codescanning).

Offensive Security

So far, we have discussed mitigations to vulnerabilities. Things like:

1. Using encrypted communication (HTTPS).
2. Password Salting & Hashing.
3. Using OAuth2.
4. Setting strict password policies.
5. Using stable node libraries.
6. Services to track and report vulnerabilities (codescanning).

Today, we'll switch our focus to **offensive security**.

Offensive Security

So far, we have discussed mitigations to vulnerabilities. Things like:

1. Using encrypted communication (HTTPS).
2. Password Salting & Hashing.
3. Using OAuth2.
4. Setting strict password policies.
5. Using stable node libraries.
6. Services to track and report vulnerabilities (codescanning).

Today, we'll switch our focus to **offensive security**.

Offensive Security is the process of attacking applications (in our case, web-based) to find and mitigate vulnerabilities.

Offensive Security

So far, we have discussed mitigations to vulnerabilities. Things like:

1. Using encrypted communication (HTTPS).
2. Password Salting & Hashing.
3. Using OAuth2.
4. Setting strict password policies.
5. Using stable node libraries.
6. Services to track and report vulnerabilities (codescanning).

Today, we'll switch our focus to **offensive security**.

Offensive Security is the process of attacking applications (in our case, web-based) to find and mitigate vulnerabilities.

Example: Until Feb 2022, Purdue's exam portal had a web-based vulnerability that allowed anyone to access student transcripts without authorization.

Defining Threat Models

The fundamental objective when setting up an attack is unauthorized access to data and / or data integrity.

Defining Threat Models

The fundamental objective when setting up an attack is unauthorized access to data and / or data integrity. For our case, this is a **vulnerability**.

Defining Threat Models

The fundamental objective when setting up an attack is unauthorized access to data and / or data integrity. For our case, this is a **vulnerability**.

Each organization has a different definition of what 'unauthorized' is.

Defining Threat Models

The fundamental objective when setting up an attack is unauthorized access to data and / or data integrity. For our case, this is a **vulnerability**.

Each organization has a different definition of what 'unauthorized' is.

For example:
Source code of Windows = Vulnerability.
Source code of Linux = Feature.

Defining Threat Models

The fundamental objective when setting up an attack is unauthorized access to data and / or data integrity. For our case, this is a **vulnerability**.

Each organization has a different definition of what 'unauthorized' is.

For example:
Source code of Windows = Vulnerability.
Source code of Linux = Feature.

This threat model may even vary between users.

Defining Threat Models

The fundamental objective when setting up an attack is unauthorized access to data and / or data integrity. For our case, this is a **vulnerability**.

Each organization has a different definition of what 'unauthorized' is.

For example:
Source code of Windows = Vulnerability.
Source code of Linux = Feature.

This threat model may even vary between users. The owner of a service having access to user data may be an issue for one user, but fine for another.

Defining Threat Models

The fundamental objective when setting up an attack is unauthorized access to data and / or data integrity. For our case, this is a **vulnerability**.

Each organization has a different definition of what 'unauthorized' is.

For example:
Source code of Windows = Vulnerability.
Source code of Linux = Feature.

This threat model may even vary between users. The owner of a service having access to user data may be an issue for one user, but fine for another.

A 'vulnerability' may not be classified as one if it doesn't breach their threat model. This distinction is the application's **security boundary**.

Setup for Web Applications Testing

To pentest web applications, we need buggy web applications.[1]

[1] too obvious?

Setup for Web Applications Testing

To pentest web applications, we need buggy web applications.[1]

For this we use **bWAPP**: `http://www.itsecgames.com/`.

[1] too obvious?

Setup for Web Applications Testing

To pentest web applications, we need buggy web applications.[1]

For this we use **bWAPP**: `http://www.itsecgames.com/`. It's also available as a VM though `bee-box`.

[1] too obvious?

Setup for Web Applications Testing

To pentest web applications, we need buggy web applications.[1]

For this we use **bWAPP**: `http://www.itsecgames.com/`. It's also available as a VM though `bee-box`. We can run it with **docker**:

```
docker pull neuralegion/beebox; docker run -p 8888:80
neuralegion/beebox
```

[1] too obvious?

Setup for Web Applications Testing

To pentest web applications, we need buggy web applications.[1]

For this we use **bWAPP**: `http://www.itsecgames.com/`. It's also available as a VM though `bee-box`. We can run it with **docker**:

```
docker pull neuralegion/beebox; docker run -p 8888:80
neuralegion/beebox
```

Next, we need to be able to manipulate packets sent to the target host. We'll use `burpsuite` for this.

[1] too obvious?

Setup for Web Applications Testing

To pentest web applications, we need buggy web applications.[1]

For this we use **bWAPP**: `http://www.itsecgames.com/`. It's also available as a VM though `bee-box`. We can run it with **docker**:

```
docker pull neuralegion/beebox; docker run -p 8888:80
neuralegion/beebox
```

Next, we need to be able to manipulate packets sent to the target host. We'll use `burpsuite` for this. Setup process:

1. Export burp's CA to your filesystem.

[1] too obvious?

Setup for Web Applications Testing

To pentest web applications, we need buggy web applications.[1]

For this we use **bWAPP**: `http://www.itsecgames.com/`. It's also available as a VM though `bee-box`. We can run it with **docker**:

```
docker pull neuralegion/beebox; docker run -p 8888:80
neuralegion/beebox
```

Next, we need to be able to manipulate packets sent to the target host. We'll use `burpsuite` for this. Setup process:

1. Export burp's CA to your filesystem.
2. Import the CA in your browser.

[1] too obvious?

Setup for Web Applications Testing

To pentest web applications, we need buggy web applications.[1]

For this we use **bWAPP**: `http://www.itsecgames.com/`. It's also available as a VM though `bee-box`. We can run it with **docker**:

```
docker pull neuralegion/beebox; docker run -p 8888:80
neuralegion/beebox
```

Next, we need to be able to manipulate packets sent to the target host. We'll use `burpsuite` for this. Setup process:

1. Export burp's CA to your filesystem.
2. Import the CA in your browser.
3. Create a proxy to the port specified in burp.

[1] too obvious?

Setup for Web Applications Testing

To pentest web applications, we need buggy web applications.[1]

For this we use **bWAPP**: `http://www.itsecgames.com/`. It's also available as a VM though `bee-box`. We can run it with **docker**:

```
docker pull neuralegion/beebox; docker run -p 8888:80
neuralegion/beebox
```

Next, we need to be able to manipulate packets sent to the target host. We'll use `burpsuite` for this. Setup process:

1. Export burp's CA to your filesystem.
2. Import the CA in your browser.
3. Create a proxy to the port specified in burp.
4. Start the interceptor.

[1] too obvious?

Outline

HTML Injection

HTML Forms allow us to supply variable input to a backend.

HTML Injection

HTML Forms allow us to supply variable input to a backend.

If this input is **reflected back to a webpage**, it effectively allows us to manipulate the page's DOM.

HTML Injection

HTML Forms allow us to supply variable input to a backend.

If this input is **reflected back to a webpage**, it effectively allows us to manipulate the page's DOM.

If proper sanitziation is not put in place, the input can be interpreted as a Javascript, which can lead to vulnerabilities like cookie stealing.

HTML Injection

HTML Forms allow us to supply variable input to a backend.

If this input is **reflected back to a webpage**, it effectively allows us to manipulate the page's DOM.

If proper sanitziation is not put in place, the input can be interpreted as a Javascript, which can lead to vulnerabilities like cookie stealing.

Let's Exploit!

If you can see this screen, I am making a mistake.

Cross Site Scripting

The previous vulnerability has a restriction; it only subsists within the single request, and is mitigated by using POST.

Cross Site Scripting

The previous vulnerability has a restriction; it only subsists within the single request, and is mitigated by using POST.

This can be undone by using stored XSS, where information is stored in the server backend and re-rendered to trigger the vulnerability.

Cross Site Scripting

The previous vulnerability has a restriction; it only subsists within the single request, and is mitigated by using POST.

This can be undone by using stored XSS, where information is stored in the server backend and re-rendered to trigger the vulnerability.

Let's Exploit!

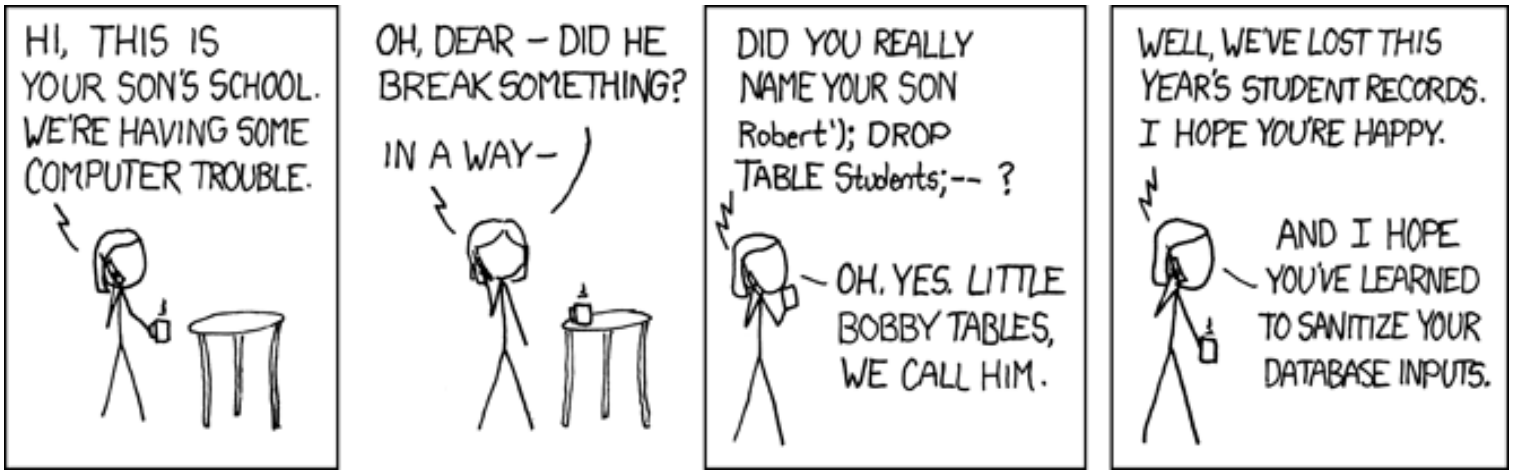
If you can see this screen, I am making a mistake.

SQL Injection

SQL backends that customize inputs take information from HTML forms.

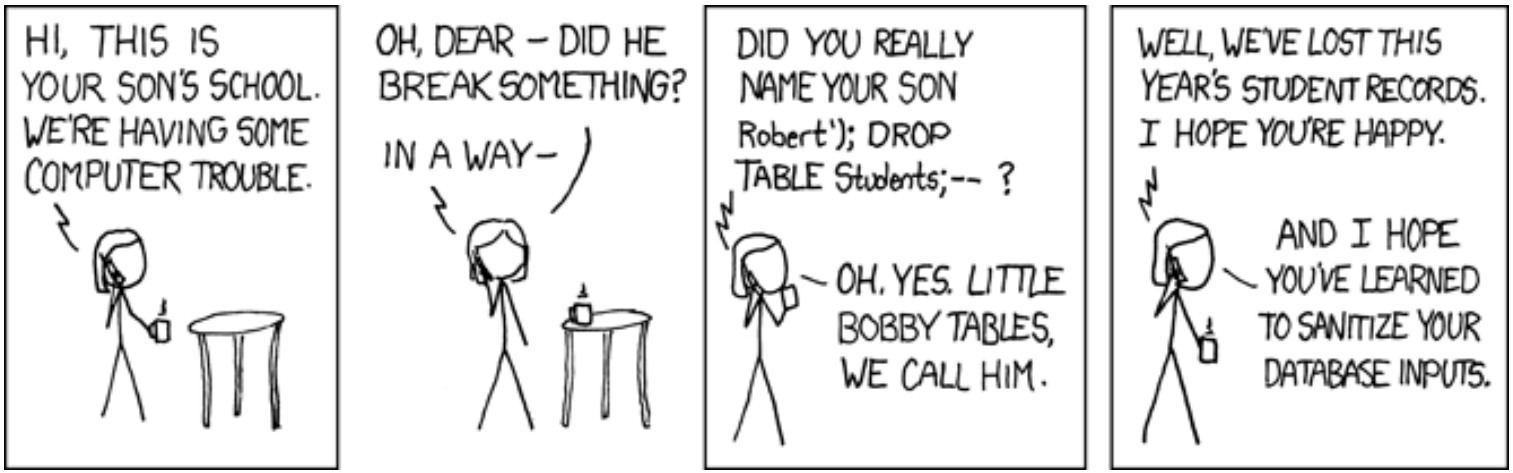
SQL Injection

SQL backends that customize inputs take information from HTML forms. Similar to XSS, if form inputs are not *sanitized*, they can be used to run aribtrary commands within the database.



SQL Injection

SQL backends that customize inputs take information from HTML forms. Similar to XSS, if form inputs are not *sanitized*, they can be used to run aribtrary commands within the database.



We can use automated approaches to find these vulnerabilities using tools like **sqlmap**.

Let's Exploit!

If you can see this screen, I am making a mistake.

Outline

Logistics

Homework #4 is due on **November 29 @ 11:59pm**.

Homework #5[2], project presentation orders[3] & instructions will be released on **November 30 @ 12:00am**.

We'll also release a submission for course evaluations, due **December 16 @ 11:59pm** for minor extra credit.

This is the last lecture-based class for the Fall 2023 semester.

[2] extra credit, optional

[3] determined randomly

Thank you!

Have an awesome rest of your day!

Slides: `https://cs.purdue.edu/homes/jsetpal/slides/offsec.pdf`

If anything's incorrect or unclear, please ping `jsetpal@purdue.edu`
I'll patch it ASAP.