Discussion Topic:

Which UML models might you use to present your chosen security vulnerability and why are they the most appropriate choice(s)?

My Initial Post

'Injection' attacks made it onto OWASP's most recent Top10 software vulnerabilities, which were decided by considering how exploitable the vulnerabilities are and how impactful those exploits could be[1]. Being somewhat familiar with SQL, I wanted to discuss SQL injections specifically.

Description of SQL Injections

OWASP links each vulnerability to Mitres' Common Weakness Enumeration entry of it, a database which gives the following description of an SQL attack:"[The use of] user-controllable inputs, [so] the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data"[2].

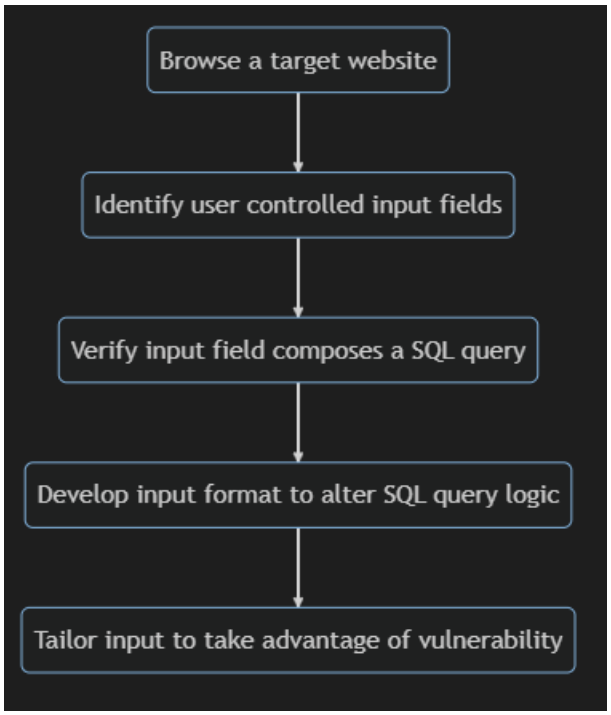
To understand the potential impact of SQL injections, it's standard to relate them to the CIA triad of ideal information assurances [3].

1. Confidentiality ~ SQL injections can facilitate the exposure of sensitive data.

2. Integrity ~ SQL injections can facilitate the alteration of data.

3. Availability - SQL injections can facilitate the destruction of data.

The following flow chart aims to show how malicious SQL injections are performed:



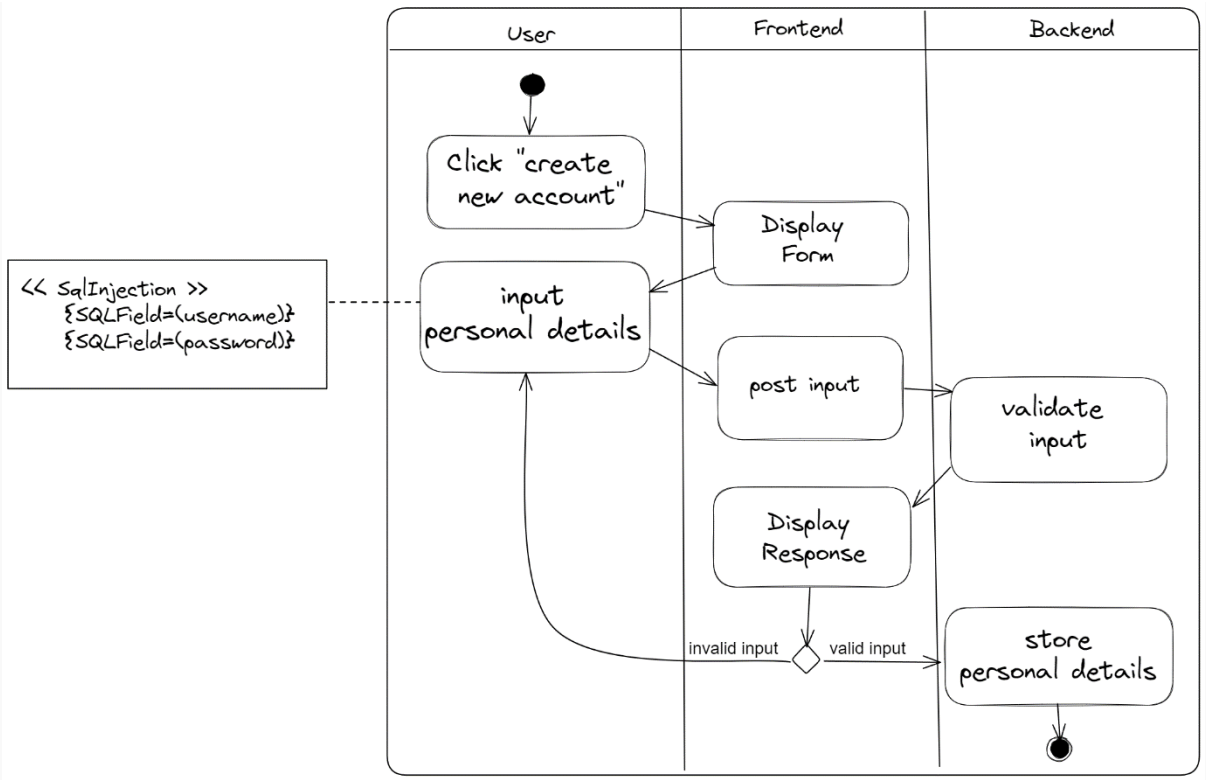
OWASP provides a theoretical example of an SQL injection[4], for the curious. Do you know of any ways to prevent SQL injections?

Model-based Security

Being accustomed to UML, it would be really convenient to be able to model security vulnerabilities using it. By modelling security vulnerabilities, we are more likely to remember to implement security protection in our software when implementing those models.

One way security proactivity is achievable, is by using UML extension entities such as stereotypes and tags. There are multiple variations of how these might be formatted, such as using the Object Constraint Language (OCL)[5], or not [6]. As OCL looks intimidating, I think it's less likely to be adopted by new developers, who are most likely to be unaware of security vulnerabilities, so I definitely favour the friendlier version.

The following activity diagram aims to be an example of a visually informative use of stereotypes and tags to indicate a security vulnerability. The stereotype and tag construction I've used is one that's been recommended [7]. Specifically, in the diagram, the stereotype is '<<SqlInjection>>' and the tags are the things written below it. The dotted lines indicate the activity that's being tagged as having a security vulnerability.



Out of all possible UML diagrams, **I think activity diagrams are the most generally useful diagrams for marking security vulnerabilities**. Some of the reasons for that include:

- The granularity of detail can be finetuned to suit either developers or other stakeholders.
- Activity diagrams are behaviour diagrams, so vulnerabilities can be situated within the flow of the software. As a developer, I would find that extra helpful.

There's definitely arguments to be made for other diagrams such as class and package diagrams, which catalogue some of the components for developers to make use of, but at this point in time, I would somewhat validate those diagrams by referring to an activity diagram. Do you find activity diagrams useful?

Software Development Lifecycles

Lastly, I wanted to bring up the task of integrating model-based security into a software development lifecycle. In a secure scrum framework, would it be beneficial to link s-tags[8] to specific activities on an activity diagram associated with security vulnerabilities, for tracking security developments?

References

[1] OWASP. (2021) OWASP TOP10. Available from: https://owasp.org/Top10/ [Accessed 4 May 2023].

[2] MITRE. (2023) CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). Available from: https://cwe.mitre.org/data/definitions/89.html [Accessed 4 May 2023].

[3] CIS. (2023) Election Security Spotlight - CIA Triad. Available from: https://www.cisecurity.org/insights/spotlight/ei-isac-cybersecurity-spotlight-cia-triad [Accessed 4 May 2023].

[4] OWASP. (2023) Testing for SQL Injection. Available from: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection [Accessed 4 May 2023]

[5] Salas, P., Krishnan, P. & Ross, K. (2007) 'Model-Based Security Vulnerability Testing', *2007 Australian Software Engineering Conference (ASWEC'07)*. Melbourne, Australia, 2007. New York: IEEE. 284-296. Available from: https://doi.org/10.1109/ASWEC.2007.31 [Accessed 4 May 2023]

[6] Jürjen, J. (2002) 'UMLsec: Extending UML for Secure Systems Development', in: Jézéquel, J., Hussmann, H. & Cook, S. (eds) *<<UML>> 2002 - The Unified Modeling Language. UML 2002. Lecture Notes in Computer Science, vol 2460*. Berlin: Springer. 412-425. Available from: https://doi.org/10.1007/3-540-45800-X_32 [Accessed 4 May 2023]

[7] Peralta, K., Orozco, A., Zorzo, A., & Oliveira, F. (2008) 'Specifying Security Aspects in UML Models', Proceedings of the Workshop on Modelling Security (MODSEC08) held as part of the *2008 International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Toulouse, France, 2008. Germany: CEUR-WS. Available from: https://ceur-ws.org/Vol-413/ [Accessed 4 May 2023]

[8] Pohl, C. & Hof, H. (2015) 'Secure Scrum: Development of Secure Software with Scrum', The Ninth International Conference on Emerging Security Information, Systems and Technologies - SECURWARE 2015. Venice, Italy, 2015. Available from: https://doi.org/10.48550/arXiv.1507.02992 [Accessed 4 May 2023]

Bibliography

OMG. (2014) About the Object Constraint Language. Specification Version 2.4. Available from: https://www.omg.org/spec/OCLF [Accessed 4 May 2023]

OMG. (2017) About the Unified Modelling Language. Specification Version 2.5.1. Available from: https://www.omg.org/spec/UML/2.5.1/About-UML#document-metadata [Accessed 4 May 2023]

My Initial Post / Tutor Response

Thank you, Brad.

Please use the Harvard approach to referencing, not the numbered approach. The Computing Department at Essex uses the Harvard scheme.

Good use of an activity diagram here, Brad. It could perhaps be extended to include an Attacker as a swimlane to show the practical steps taken when a breach is happening.

I note entries in the Reference list which are not cited inline e.g., Peralta et al. (2008). If this source has informed your reading but has not been used to support your writing, please move it in the Bibliography instead.

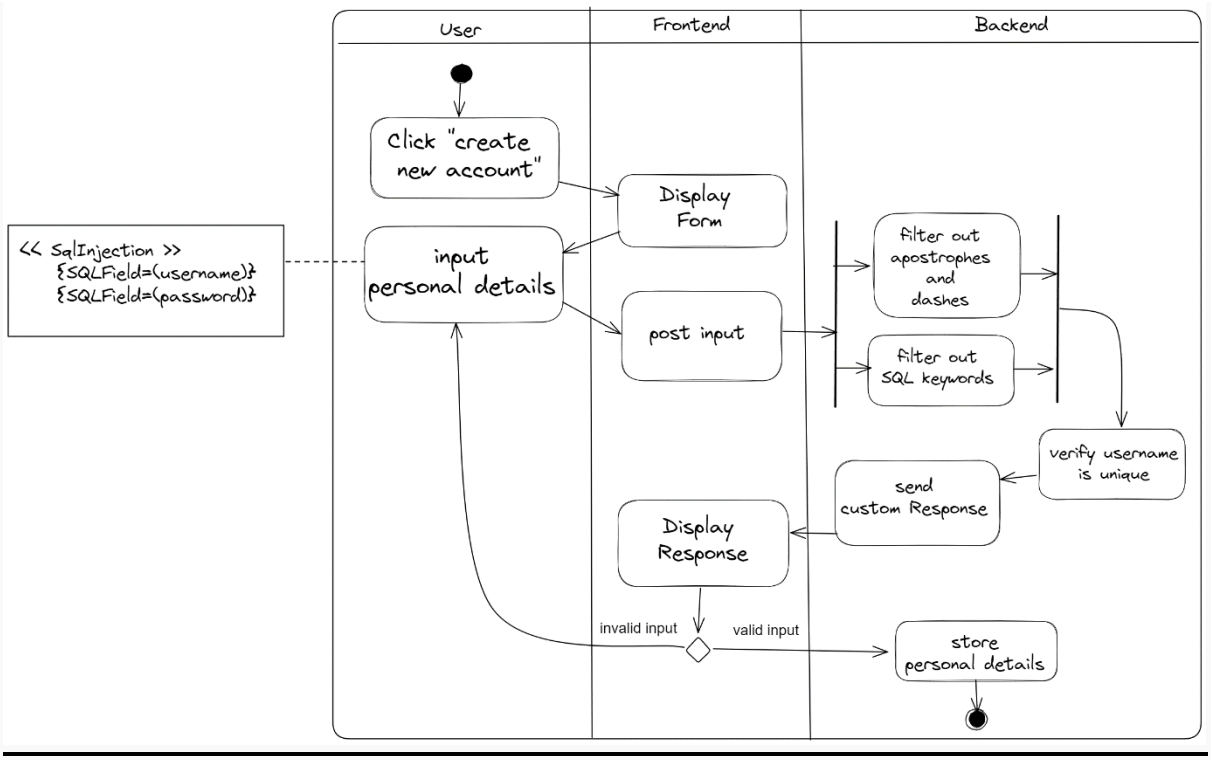
Best wishes,
Cathryn

My Initial Post / Tutor Response / My Response

Thanks for taking the time to read and critique this.

After reading your response I've realised that the 7th link goes to a collection of papers and not the paper itself. The correct link is this one: https://ceur-ws.org/Vol-413/paper11.pdf. Information was taken from that paper directly, as specified by the sentence: "The stereotype and tag construction I've used is one that's been recommended [7]". That paper specifies a stereotype and tag construction that I've made use of in the activity diagram. If I'd have used Harvard's approach for inline citations, I hope that would have been clearer, and I've regretfully missed an opportunity to practise that.

Regarding the suggestion to incorporate more details about breaches into the activity diagram; I would like to build upon that with the note that **designing security solutions is more useful for transitioning into implementation, than just annotating the security vulnerabilities of a design**. Annotations of the kind I demonstrated, are only useful in a multi-stage design phase, where solutions are made afterwards. Therefore, drawing from 'CAPEC-66: SQL Injection' entry of MITRE's Common Attack Pattern Enumeration and Classification (CAPEC) list of mitigations for SQL injections; I would like to share a revision of the activity diagram:



I couldn't guarantee that an outside stakeholder would be able to tell which activities constitute mitigations against SQL attacks, so I think that's a potential weakness of this diagram. As I designed this diagram, I know that the mitigations are the following activities: 'filter out apostrophes and dashes', 'filter out SQL keywords' and 'send custom Response', but I wonder what the most effective technique for clearly marking those activities as security mitigations is? Possible techniques I guess would be: the use of more specialised swim lanes, the addition of events into the diagram, or perhaps more varieties of stereotype. I'm curious about opinions on that.

References

Mitre. (2021) CAPEC-66 : SQL Injection. Available from: https://capec.mitre.org/data/definitions/66.html [Accessed 11 May 2023].

Peer A Initial Post

A06:2021-Vulnerable and Outdated Components

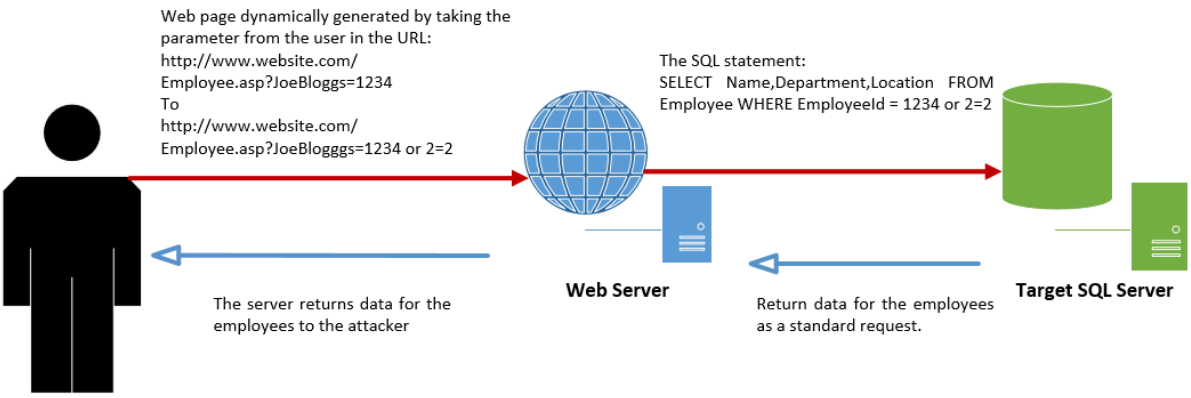
Vulnerable and Outdated Components has moved up from ninth to sixth in the OWASP Top 10 Web Application Security Risks. Having worked for many years in education and now in a Government Department I have witnessed these organisations sweating out the use of hardware and software in order to keep costs down. This can lead to components becoming vulnerable to attacks or being out of date.

Java-script libraries are used by web developers in order to make sites more functional, however this can be open to attacks if not kept up-to-date and result in

the site being potentially exposed **(**Lauinger et al, 2018). According to Tang et al (2015), you are able to predict these vulnerabilities through text mining or software metrics, with software metrics seen as the most cost-effective.

Another area where this is an issues is through SQL injection attacks, this can allow sensitive data to be updated or read as well as running commands and accessing files from the servers (Guimarães, 2009). An example of this can be seen in Figure 1.

Figure 1
SQL Injection Attack



The Internet of Things (IoT) is seen as a revolution, however this can be susceptible to attacks due to insecure software configuration (Jiang, Lora, and Chattopadhyay, 2020). Old/unpatched dependencies in the dependency chain of the components being used. There is also an issue with IOT that older dependencies or those that are unpatched can be exploited by cybercriminals. This is particularly an issue when industries use this technology to gain efficiencies, for instance the Amazon Ring was subject to an attack, whereby the hackers could get live streams from the systems through weak, recycled and default identifications (Smith, 2020).

References:

B. D. A. Guimarães, Advanced SQL injection to operating system full control, Black Hat Europe, white paper, 2009.

Jiang, X., Lora, M. and Chattopadhyay, S., 2020. An experimental analysis of security vulnerabilities in industrial IoT devices. ACM Transactions on Internet Technology (TOIT), 20(2), pp.1-24.

Lauinger, T., Chaabane, A., Arshad, S., Robertson, W., Wilson, C. and Kirda, E., 2018. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. arXiv preprint arXiv:1811.00918.

Omolara, A.E., Alabdulatif, A., Abiodun, O.I., Alawida, M., Alabdulatif, A. and Arshad, H., 2022. The internet of things security: A survey encompassing unexplored areas and new insights. Computers & Security, 112, p.102494.

Sengupta, J., Ruj, S. and Bit, S.D., 2020. A comprehensive survey on attacks, security issues and blockchain solutions for IoT and IIoT. Journal of Network and Computer Applications, 149, p.102481.

Smith, S.W., 2020. Securing the Internet of Things: An Ongoing Challenge. Computer, 53(6), pp.62-66.

Tang, Y., Zhao, F., Yang, Y., Lu, H., Zhou, Y. and Xu, B., 2015, August. Predicting vulnerable components via text mining or software metrics? An effort-aware perspective. In 2015 IEEE International Conference on Software Quality, Reliability and Security (pp. 27-36). IEEE.

https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

Peer A Initial Post / My Response

Hi,
Thanks the interesting read. I like the fact that you chose to talk about 'vulnerable and outdated components', because it's a good example of a security vulnerability that UML doesn't have a prescribed view to model it's solution. UML has a package view to show dependencies between packages, but Rumbaugh et al (2005), the pioneers of UML, state that versioning is outside the scope of it.

Does package management software, such as pip for python, makes it more or less likely that software designers take an active role in package management? Package management software can give warnings when vulnerabilities are detected,
but studies such as Athalye et al's (2014) have shown that package management software can itself be an attack vector, in man the middle attacks for example. I'm curious if you have any input on the mitigation of the security vulnerability associated with outdated components?

References

Athalye, A, Hristov, R., Nguyen, T. & Nguyen, Q. (2014) *Package Manager Security*. Available
from: https://pdfs.semanticscholar.org/d398/d240e916079e418b77ebb4b3730d7e959b15.pdf
[Accessed 11 May 2023].

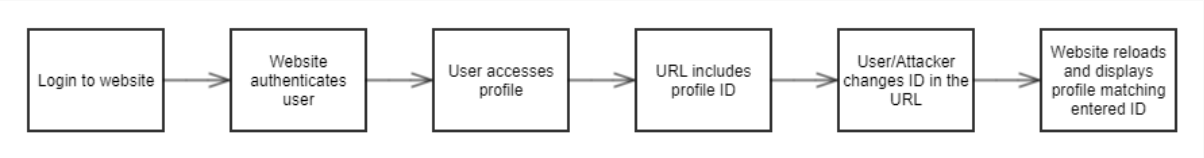
Rumbaugh, J., Jacobson, I. & Booch, G. (2005) *The Unified Modeling Language Reference Manual*. 2nd ed. Boston, USA: Pearson Education, Inc.

Peer B Initial Post

The focus of my initial post is the "Broken Access Control" vulnerability, which appears to be a more common issue than one may think. According to OWASP, 38

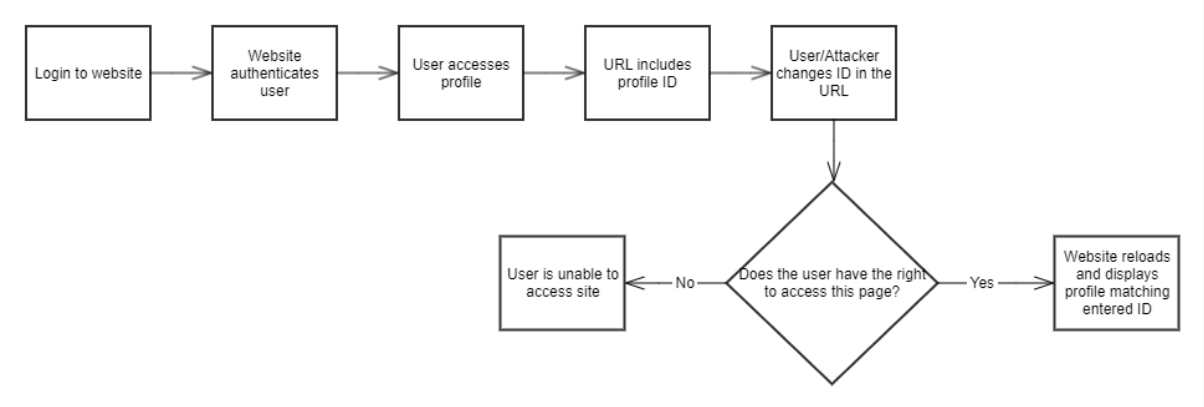
out of 100 websites suffer from this vulnerability (OWASP, 2021). Ranking fifth only a few years back, "Broken Access Control" has moved up to be the number one vulnerability in the most recent edition of the OWASP top ten.

Per definition, "Broken Access Control" refers to a faulty access control system which allows an attacker to access data outside of their intended permission. In its worst form, it may allow an attacker to modify and delete data or take control over an entire system (OWASP, 2021). This may consequently lead to financial losses, user data being compromised, and damage to the reputation of the affected company (Hassan et.al., 2018).



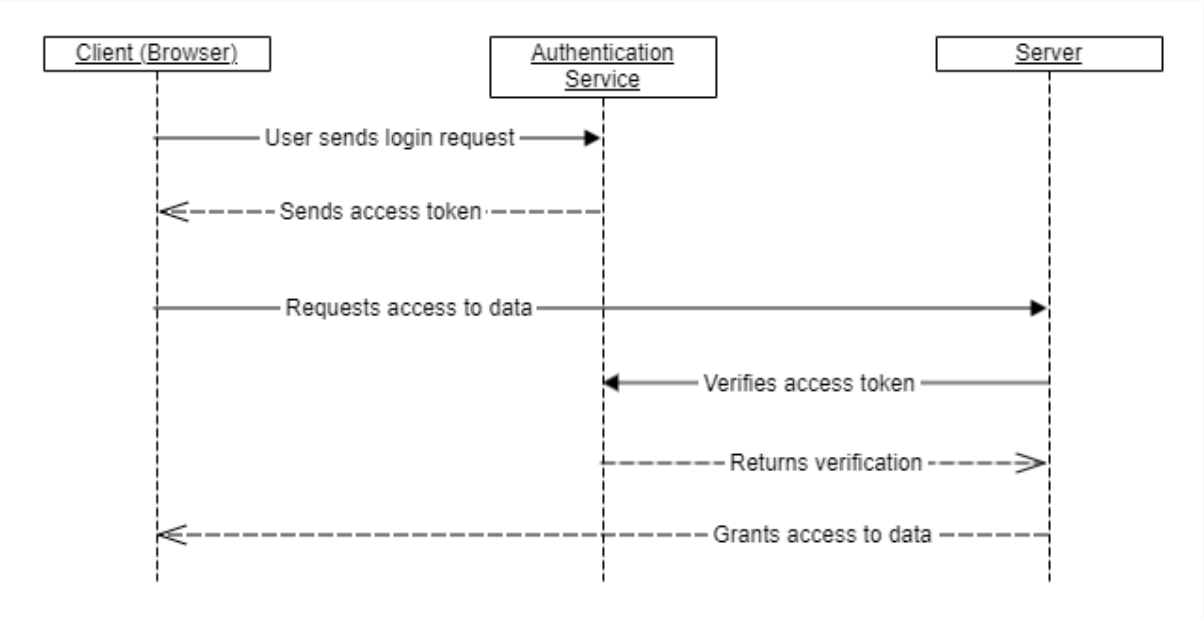
As described in the flowchart above, one of the ways for this vulnerability to occur is that the website includes a user's profile ID in the URL. It would be relatively safe to assume that changing this ID by one integer up and down would result in the profile ID of another user. If the website does not check for privileges or does not re-authenticate a user when one of its pages is accessed directly via a link, then this may result in unauthorized access to another person's profile.

Therefore, a user should be authenticated each time they try to access another page. If the user attempts to retrieve information they don't have the privileges for, then access should not be granted (see flowchart below).



This concept is best represented with a UML Sequence Diagram, as it shows the transmission and exchange of messages between the server(s) and the client to authenticate a user and only grant access to those resources the user has privileges

to.



In this example, I have decided not to represent the "actor" (i.e. the user), as all of the valuable interaction takes place between objects. When the user attempts to log in to their account, the client, which in this case is a web browser, sends a login request to the authentication service. The authentication service verifies the user's credentials and returns an access token, which is stored on both the server and the client.

When the user requests to access data on the server, the server will attempt to verify the access token by communicating directly with the authentication service. If the authentication service deems the access token to be valid, the server will grant access to the requested data. If the access token is invalid or the user fails to produce valid login credentials altogether, then the authentication service and server will both reject the client's attempts to access data. As the server will always communicate with the authentication service whenever a new page or new data is accessed, an attacker will be unable to simply change URLs or access pages directly via a link to gain unauthorized access.

References

OWASP (2021) A01:2021 - Broken Access Control. Available from: **https://owasp.org/Top10/A01_2021-Broken_Access_Control/** [Accessed 6 March 2023].

Hassan, M. M., Ali, M. A., Bhuiyan, T., Sharif, M. H. & Biswas, S. (2018) 'Quantitative Assessment on Broken Access Control Vulnerability in Web Applications', *International Conference on Cyber Security and Computer Science*

(ICONCS'18) Safranbolu, Turkey, 18-20 October. Available from: https://www.researchgate.net/profile/Saikat-Biswas-10 [Accessed 6 May 2023].

Peer B Initial Post / My Response

Hi,
I really like the way you've separated the topic into two flow diagrams, the first to introduce the security vulnerability, and the second to introduce the security mitigation. It's an excellent use visual scaffolding to clarify the topic, and to me, it demonstrates criticality, through the identification, and then development of key information.

I also really liked your sequence diagram, especially the inclusion of objects that exist in a service-oriented architecture, because software development often exists within the context of a business, so I think it's a generally appropriate architecture to use. I'm not experienced in service-oriented architecture, so I appreciate seeing it in your example. Do you by any chance know of any good libraries in Python for building an authentication service?

Peer B Initial Post / My Response / Peer B Response

Hi Brad,

Thanks for your comment! The only Python library I have used to experiment with user and password authentication is called 'bcrypt'. The 'bcrypt' library allows you to hash passwords to store them securely. You can then use these hashed passwords to authenticate login attempts. The 'bcrypt' library also uses something called 'salt', a random value that is added to the user's password when it is hashed. This guarantees that the hash is unique, even if two users use the same password.

Other than that, I have looked at libraries such as "Authlib", which allows you to implement authentication protocols, but I have not had the chance to use it yet.