Final Project Report

Website Security Research

June 2nd 2021

Team Members

Patrick Dougan
Reed Hardin
Isfandyar Rabbani
Michael Volz

Project URL: http://ec2-34-210-43-2.us-west-2.compute.amazonaws.com:3000/

Introduction

As the quarter comes to a close, so does our research project on website security. At the beginning, our research group embarked on a mission to create a weak and a strong version of a useful website that could provide contrasting examples of security elements implemented into a system as a whole. Once knee-deep in the waters of vulnerability exploitation, we found that making a single system would prove too complex to scale.

As a result, our team adapted and found novel ways to demonstrate weaknesses in the attack surface and how to exploit them. We focused on implementing separate weak features, rather than one system as a whole, and demonstrating how one would go about alleviating the weakness and what programming principles come into play when developing a rock-solid web application.

Our group has developed a series of deliberately insecure web applications which are vulnerable to several common web attacks, found on the OWASP Top 10 Web Application Security Risks list. Our efforts on this project have focused on learning about specific vulnerabilities and threats to web site application servers. We have learned how to implement mechanisms to exercise these threats and expose vulnerabilities, as well as how to thwart these threats. We have documented each of the vulnerabilities found in the web applications, along with how to prevent them.

Our team GitHub repository includes all the source code pertaining to each specific feature, and our simple index page contains guides on how to use this source code to learn about brute force dictionary attacks on password hashes, cross-site scripting, SQL injections into a simple search function, database security role vulnerabilities, and sensitive data exposure.

User Perspective

Our primary "user-focused" deliverables include a main index page which links to the documentation on each of the attacks that was implemented, including a demonstration which links to a deliberately vulnerable web application. We also have a GitHub repo with all the source code used to create this project and the documentation.

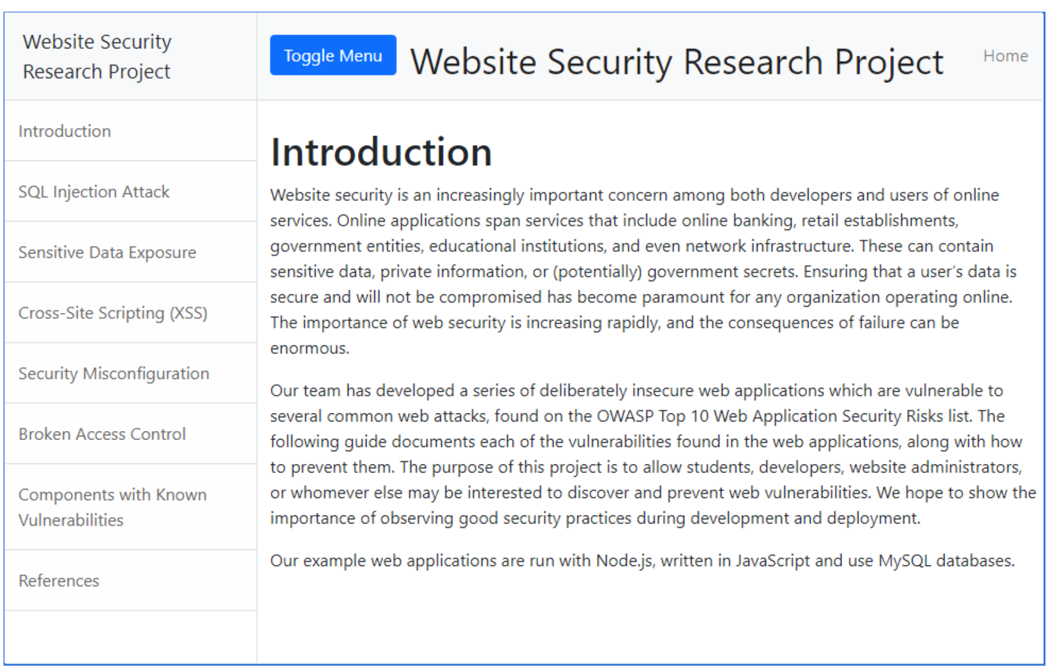
For the demonstration, the user is able to use each particular vulnerable web application to experience first-hand how various vulnerabilities can exist and how they can be exploited by an attack (basically to learn how to not make a website). The user gets a chance to wear a white hat for a moment to get into the mind of an adversary.

Conversely, for some of the attacks, the user can use a 'hardened' version of a web application to experience how to create a web application that is secure, minimal in attack surfaces, and considerably future-proof to unknown threats. This perspective will hopefully help the user fully comprehend just how perforated various attack surfaces can be and how some of these exploits can be very easily fixed or prevented.

User Instructions

1. First, you will want to navigate to our main index website, which is located at: http://ec2-34-210-43-2.us-west-2.compute.amazonaws.com:3000/

2. You will see an introduction to our project, along with a side menu which links to each exploit we implemented:



3. You can click through and explore each of the attacks as you please. Each attack has a demonstration section which links to an external, deliberately vulnerable web application to showcase the exploit first hand. Some attacks also have a fixed version of the web application where the exploit is no longer possible.

Introduction

Cross-Site Scripting (XSS) attacks are a form of injection vulnerability, where malicious scripts are injected into trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code to the end user. Flaws that allow these attacks to succeed can occur anywhere a web application uses input from a user within the output it generates without validating or encoding. Because the user has no way to know the script should not be trusted it will execute the script. The malicious script will be able to access any sensitive information being used by the site.

Scenario

Vulnerable [Link]
Secured [Link]

This application allows users to post their name email and a comment. It then stores these fields in an SQL database.

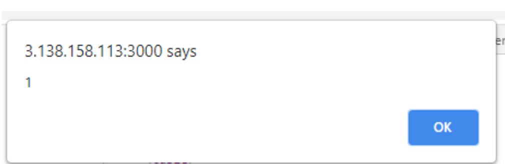
```
app.post('/', urlencodedParser, function(req, res) { var body = req.body; var sql = "INSERT INTO comments (name, email, comment) VALUES
('"+body.name+"','"+body.email+"','"+body.comment+"')";
```

The application then redirects to the original page and loads all comments. However the comment section is not escaped when the page is rendered.

```
app.get('/', function(req, res) { var sql = "SELECT * FROM comments" con.query(sql, function(err, result) { if (err) throw err;
res.setHeader('Content-Type', 'text/html') res.render('index', {results: result}) });
```

Demonstration

If we submit a comment with the following code: <script>alert(1)</script> This will store a comment in the SQL DB. When the page is refreshed it will load a webpage and will process the alert(1) message as javascript.



Prevention

One of the key ways to mitigate XSS is to properly escape characters. If our sample comment was properly escaped then the webpage would display <script>alert(1)</script> as a string instead of executing as a JS function.

4. You may also navigate to our GitHub repo to check out the source code used to make all of the web applications and the index application at the following URL: https://github.com/PatrickDougan/Website-Security-Research-Project

5. Test, test, test! Send a request to the server using postman or interact using the UI for the feature in question, and really find out whether the weak and strong versions of the website are vulnerable to the attack.

Languages, Frameworks, Libraries, etc

***Node.js*/Express** - Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js was used in each web application along with the main index website.

JSON Web Token - JWT uses a Javascript library that applies an encryption algorithm to create a time-sensitive token that does not require a private and public key to communicate.

JavaScript (JS) - JS is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. JS was used to create the functions and routes in each web application.

Embedded Javascript (EJS) - EJS is a view engine with a simple templating language that lets you generate HTML markup with plain JavaScript. EJS was used for the creation of the main index website, along with the views for the SQLi attack and broken access control exploit.

MySQL - MySQL is an open-source relational database management system. MySQL was used in the creation of test databases which are used in each web application.

HTML/CSS/Bootstrap/jQuery/Handlebars - These are various, well-known tools and libraries which aided in the creation of most of our web applications, along with the templates for presenting the write-ups for each attack.

Augustine / http-server - These two npm packages allow us to serve pdf files from our website. These are used in the creation of the component vulnerability.

Team Member Contributions

As a team we originally planned to spend three weeks developing two or more “test attack” websites (each will include authentication, database, and a database interface). We then expected to shift into research and coding each attack individually, with the ultimate goal being to demonstrate eight total attack types. For the most part, we actually were able to stick with this plan.

Patrick Dougan

During the initial two weeks of our project development, Patrick did a great deal of research and investigation into alternative methods to demonstrate vulnerabilities (including using the Heroku and the nodegoat platform). When that was complete (roughly the end of week4) he shifted to investigating the Cross-Site Scripting (XSS) attack, researching it, creating a meaningful demonstration, and a final summary. When that was complete, Pat shifted to researching XXE vulnerabilities. Unfortunately, XML parsers no longer support the external entities which is what made XXE possible. From there he moved to Components with Known Vulnerabilities, concluding with a summary and a demonstration.

Reed Hardin

Our initial few weeks of development saw Reed exploring alternative methods to demonstrate vulnerabilities (including using the Heroku and other platform demonstrations). He researched pre-made vulnerable websites for ideas for the team and to see how easily they could be implemented in our own project. When that was complete (roughly the end of week4) he shifted to investigating the SQL Injection attack, researching current methods used, creating a meaningful demonstration, and a final summary. He also created a Bootstrap write-up template so each team member's findings would be uniform. When that was complete, Reed shifted to researching Broken Access Control vulnerability, wrapping everything up with a demonstration and summary.

Isfandyar Rabbani

For the first two weeks, Fandi researched and implemented a handful of websites on the AWS platform, namely EC2, using a NodeJS framework and a local MySQL database. The deployment of this app was a useful example for implementations for the rest of the group. For the next few weeks, Fandi implemented simple authentication, then working up to 256-bit encryption using Passport and JSON Web Token. Finally, Fandi spent the later part of the quarter researching brute force dictionary attacks and integrating it into the project. Fandi also served as our point of contact with our TA and with OSU's IT department at various points throughout the quarter.

Michael Volz

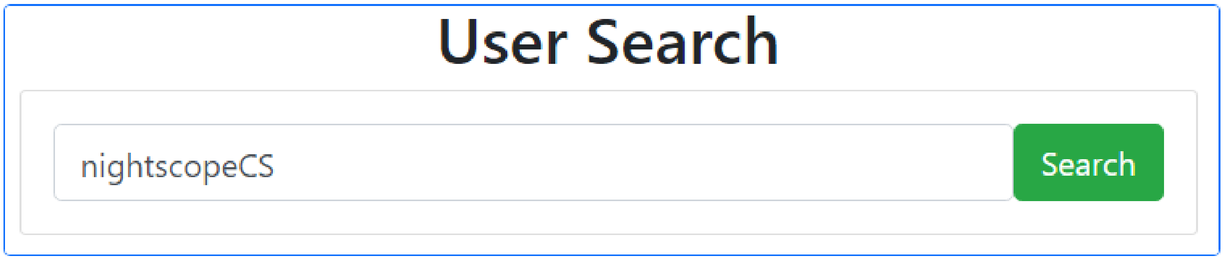
Initially, Mike started by developing a "stand-alone" website, hosted on a domain that he owns; this is a simple setup using Node.js, handlebars, express, and a mySQL database. The end result was two functional sites (each appears very similar, but is distinctly different "under the hood") with two distinct databases. It turned out that accessing a mySQL database on the AWS platform was much easier than one hosted on the privately-owned domain, so that part of the plan shifted slightly: each mySQL database is currently defined and hosted on AWS. Mike then shifted to the Sensitive Data exposure for two weeks, and used the simple database to demonstrate that particular vulnerability and authored a summary. Finally, Mike researched and created a simple demonstration for the Security Misconfiguration exposure.

Task-by-Week Table for Each Team Member:

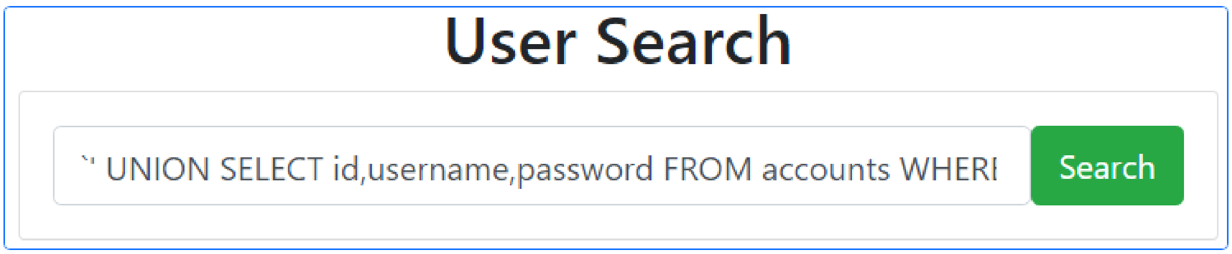
	Team Member - Task	Time (hours)
Week 3	Patrick: Support website development Reed: Research vulnerable websites Michael: Start building stand-alone website (domain) Isfandyar: Start building OSU-based website	12 13 14 13
Week 4	Patrick: Support website development Reed: Support website development Michael: cont. building stand-alone website (domain) Isfandyar: cont. building OSU-based website	13 12 14 13
Week 5	Patrick: Support website development Reed: Support website development Michael: Finish up stand-alone website Isfandyar: Finish up OSU-based website	12 14 13 14
Week 6	Patrick: Start work on cross-site scripting (XSS) Reed: Start work on SQL injection attack Michael: Start work on sensitive data exposure Isfandyar: Start work on broken authentication	14 13 12 12
Week 7	Patrick: Wrap up cross-site scripting (XSS) Reed: Wrap up SQL injection attack Michael: Wrap up sensitive data exposure Isfandyar: Wrap up broken authentication	13 12 14 14
Week 8	Patrick: Start work on known components with vulns Reed: Start work on broken access control Michael: Start work on security misconfiguration Isfandyar: Start work on password cracking (stretch)	12 15 14 13
Week 9	Patrick: wrap up known components with vulns Reed: Wrap up broken access control Michael: Wrap up security misconfiguration Isfandyar: Wrap up password cracking (stretch)	13 11 12 12
Week 10	Patrick: Final Report Input, Demo Testing Reed: Final Report Input, Demo Testing Michael: Final Report Input, Demo Testing Isfandyar: Final Report Input, Demo Testing	12 12 12 12

Graphical Examples

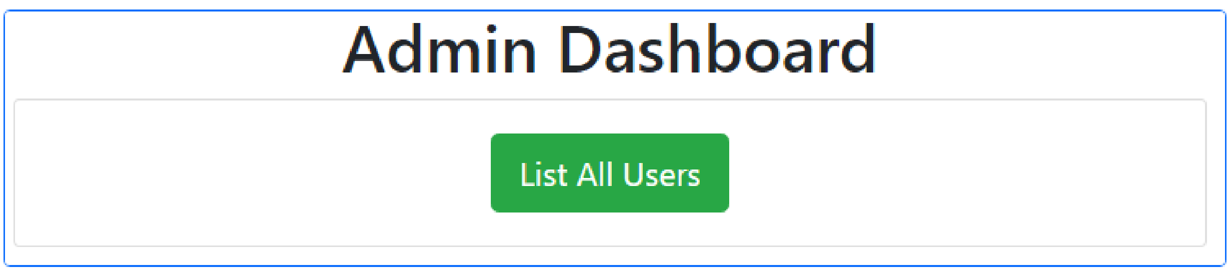
User Search Function



SQL Injection Attack via User Search Function



Admin Dashboard with Admin Functions via Forced Browsing (Broken Access Control)



Sensitive Data Exposure demonstration:

(sensitive data exposed)

Student Name	Student ID	Grade	Band	Modify Student
Hill Billy	123456782	10	Beginning	Delete Update
Chuck Bryant	541022090	8	Intermediate	Delete Update
Josh Clark	295461994	5	Beginning	Delete Update
Sly Conway	452244379	6	Intermediate	Delete Update
Steve Dubner	525989535	7	Advanced	Delete Update
Emma Green	133381551	11	Advanced	Delete Update
Hank Green	267796427	5	Beginning	Delete Update
John Green	263674616	7	Advanced	Delete Update
Terry Gross	3219592	8	Advanced	Delete Update
Brady Haran	467560585	8	Advanced	Delete Update
Chris Hardwick	505485752	7	Advanced	Delete Update
James Holden	520152851	8	Advanced	Delete Update
Henry Jameson	520158072	6	Intermediate	Delete Update
Jessica Jiang	4379507	6	Intermediate	Delete Update
Joshua Johnson	585657333	7	Intermediate	Delete Update
Billy Jones	434557721	5	Beginning	Delete Update
Alex Kamal	578796715	7	Intermediate	Delete Update
Sarah Koenig	539203664	5	Beginning	Delete Update
Bobby Smith	479602656	6	Advanced	Delete Update
Robert Smith	502439747	8	Advanced	Delete Update

(sensitive data is encrypted)

Student Name	Student ID	Grade	Band	Modify Student
Chuck Bryant	whIyOlmiXAZWj47nonUsbw	8	Intermediate	Delete Update
Josh Clark	MYtazc42mBMRj5djgsUYmw	5	Beginning	Delete Update
Sly Conway	eEl2OGpGBlMkrlrfet5E9Q	6	Intermediate	Delete Update
Steve Dubner	cHpcmiDeu4F9wF4Ic06KBw	7	Advanced	Delete Update
Hank Green	Iej1iinGBu7MLhQtUtO0sw	5	Beginning	Delete Update
John Green	mPQ6R/jfUwEP/ESU61KJtw	7	Advanced	Delete Update
Terry Gross	lNmudRqrPi2UVMZ+5VQb4A	8	Advanced	Delete Update
Brady Haran	weWhrCQXkNREfRpEIGNuqA	8	Advanced	Delete Update
Chris Hardwick	tGmaTajRAhq9DtO+kOBmAA	7	Advanced	Delete Update
Henry Jameson	jp09TB5n4CA3vdH10aCglw	6	Intermediate	Delete Update
Jessica Jiang	z9c6e64j1W6Zyy+qsOwjYA	6	Intermediate	Delete Update
Joshua Johnson	5XdMiDo/9qiI7OWwjHTi1g	7	Intermediate	Delete Update
Billy Jones	xlx15HFMjxH2sAU2ywNHGg	5	Beginning	Delete Update
Sarah Koenig	FXyCb18JQaO327gY6aPBJg	5	Beginning	Delete Update
Bobby Smith	KMFdQto8YBuS5ZHviou9zQ	6	Advanced	Delete Update
Robert Smith	vd3X/Rt/TCIQSNeG5YLj7g	8	Advanced	Delete Update
Frank Zip	m0YedkPdhPe5dPCieeBR+A	7	Beginning	Delete Update

Deviations from Original Plan

Our original plan was to develop a basic web application. Our website was supposed have a login functionality. It would have had a database table for authentication and a second table that contains the rest of the user's information. Once logged in, the user would have access to create, read, update, and delete users. In practicality, we ended up developing separate features rather than a cohesive application.

As mentioned in the "Team Member Contributions" sections above, we were able to maintain our overall project schedule. We spent roughly two weeks creating multiple "attack" sites using various types of infrastructure and methods. We were able to demonstrate some simple authentication pages, and a few database interfaces before moving forward.

At that point (roughly class week 5) we shifted to investigating attacks. Each team member focused on an individual attack, and did their best to both understand it and create some sort of demonstration for that attack. Most of the team then (during week 7) shifted to a second attack, while one team member focused on the stretch goal (password cracking).

The overall schedule was adhered to by the team. Our biggest deviation was exactly which attack each individual team member worked on. We ended up shuffling a bit based on individual interest in specific attacks, and ability to complete a summary and a demonstration (at the end of the process).

Research Goals

We understand that web applications are one of the most targeted systems by malicious hackers, representing the largest attack surface available and serving as points of entry for many attacks. This is due to the fact that web applications are (by definitions) the simplest point of access for organizations, companies, and governments. To secure web applications, penetration testers will try different attack vectors to ensure there are no vulnerabilities. Any vulnerabilities found will then be corrected, patched, or disabled.

Our goals for this project were to create different versions of vulnerable web applications. Once we have that completed, we developed attack methodologies based on the OWASP top 10 vulnerabilities. One notable prior approach we noted is the DVWA http://www.dvwa.co.uk/ . DVWA is a vulnerable web application designed for beginner penetration testers. In addition to the application there are also many write ups on attacks against DVWA. These served as good starting points for our development of our attacks against our vulnerable web application.

Experimental Approaches

As noted above, we started by developing a series of basic web applications with a variety of functionality, including authentication ,database access, and other useful functions. At that point, each team member researched different web app vulnerabilities listed on OWASP, exposed a particular vulnerability within at least one of the sites, and then created an attack to demonstrate the vulnerability. Finally, we created a series of summaries (one for each vulnerability) highlighting each successful attack and the methodologies used to achieve them.

Experimental Setup

A series of relatively simple web applications which are used as the “targets” for testing. The web applications are hosted separately, allowing each individual team member to work on their assigned exploits in convenience and as needed. Also, some of these applications had to be developed differently to adequately emphasize the attack demonstration. Development of a main index website which directs the end-user to each exploit. Maintenance of a central GitHub repository with the source code to each exploit and fixes.

Some of the features included in these web applications include:

- User input (such as a search function)
- Various vulnerable databases with “sensitive” data
- CRUD functionality in some scenarios
- “Hardened” or fixed versions of vulnerable apps for some attacks
- Detailed documentation of each attack and how to prevent them

Conclusion

Our group spent over a combined 400 hours researching and reverse engineering common exploits found in web applications, as well as designing, implementing, and deploying website instances that were used to demonstrate and test the various vulnerabilities researched. We had mixed findings regarding the complexities of defensive solutions to common vulnerabilities. While some attacks are easier to implement than others, defensive strategies are all equally difficult to implement.

Defensive programming strategies require a holistic approach to programming; One that does not compromise on any aspect of the program simply for convenience or limited time for development. When web applications and the data they protect are programmed from the very beginning with a realistic idea of what threats lurk beneath at every step, they can truly be safe to use for the user.

In today's world of unending features and lack of privacy, it is impossible to predict every single attack surface in existence. This is why the design and implementations of various security principles should be built into the program since its inception. Some of our group members found that working from the ground up in features was too difficult to integrate with important security features, such as password encryption. It was found that it would have been far easier to first implement an encryption algorithm and verify an output, and then work around that to create the rest of the features of the application. This is due to the fact that the data is easier to track and debug when you are a hundred percent positive where it is going all the time.

As a team, we are very pleased with our final product, which is a collection of deliberately insecure web applications. We have exposed how some of the most common web attacks are facilitated. We have created documentation which includes a demonstration of each particular exploit, along with how to fix and prevent the issue from occurring in the end-user’s own applications. Our goal is to show the importance of observing good security practices during development by demonstrating the potential for exploitation when an application is not hardened against these common attack vectors.

References

General

https://owasp.org/www-project-top-ten/
https://owasp.org/www-project-vulnerable-web-applications-directory/
https://www.w3schools.com/w3css/w3css_templates.asp
https://www.secsignal.org/en/news/how-i-hacked-a-whole-ec2-network-during-a-penetration-test

Brute Force Attack

https://en.wikipedia.org/wiki/Dictionary_attack
https://www.sitepoint.com/using-json-web-tokens-node-js/
https://github.com/auth0/express-jwt

SQL Injection Attack

https://owasp.org/www-project-top-ten/2017/A1_2017-Injection
https://portswigger.net/web-security/sql-injection
https://www.neuralegion.com/blog/sql-injection-attack/
https://www.rapid7.com/fundamentals/sql-injection-attacks/
https://www.guru99.com/learn-sql-injection-with-practical-example.html
https://www.veracode.com/blog/secure-development/how-prevent-sql-injection-nodejs

Sensitive Data Exposure

https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure
https://hdivsecurity.com/owasp-sensitive-data-exposure
https://blog.detectify.com/2016/07/01/owasp-top-10-sensitive-data-exposure-6/
https://medium.com/shallvhack/owasp-sensitive-data-exposure-attacks-7ef41e6b4a59
https://www.gosolis.com/blog/has-your-organization-been-breached-by-solar-winds-malware/
http://index-of.co.uk/Cloud-Technology/Node.js%20Recipes%20-%20Cory%20Gackenheimer.pdf

Cross-Site Scripting (XSS)

https://owasp.org/www-community/attacks/xss/

Components with Known Vulnerabilities

https://luke0922.gitbooks.io/owsap-top-10-scurity-problem/content/chapter9.html
https://www.siemba.io/post/owasp-top-10-using-components-with-known-vulnerabilities
https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities

Security Misconfiguration

https://www.giac.org/paper/gsec/317/default-password-threat/100889
https://outpost24.com/blog/What-are-security-misconfigurations-and-how-to-prevent-them
https://www.acunetix.com/blog/web-security-zone/security-misconfigurations/
https://www.thesslstore.com/blog/how-to-prevent-security-misconfiguration/
https://www.computerworld.com/article/2716804/if-your-router-is-still-using-the-default-password--change-it-now-.html
https://www.domaintools.com/resources/blog/unraveling-network-infrastructure-linked-to-the-solarwinds-hack#

Password Cracking

https://en.wikipedia.org/wiki/Password_cracking
https://www.sohamkamani.com/blog/javascript/2019-03-29-node-jwt-authentication/
https://github.com/lmammino/jwt-cracker
https://github.com/royanguiano/Node-cracking-password-app

Broken Access Control

https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control
https://avatao.com/blog-broken-access-control/
https://cwe.mitre.org/data/definitions/425.html
https://hdivsecurity.com/owasp-broken-access-control
https://www.immuniweb.com/blog/OWASP-broken-access-control.html
https://www.veracode.com/security/failure-restrict-url-access