CS547 Project Update 2

Team Members:

Ravi Kishan	1701CS39
Vivek Kumar Shaw	1701CS55
Shrish Chandra Sharma	1701CS58

Tools for vulnerability detection

We have done the following analysis of tools and techniques used for vulnerability detection.

Analyse Code Smells :-

Code smells are usually not bugs—they are not technically incorrect and do not currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future.

Many interesting tools exist to detect bugs in our C++ code base like cppcheck, clang-tidy and visual studio analyzer.

But what about the detection of the bug-prone situations?
We have used CQLinq queries to do the analysis :

CppDepend provides a code query language named CQLinq to query the code base like a database. Developers, designers and architects could define their custom queries to find easily the bug-prone situations.
With CQlinq we can combine the data from the code metrics, dependencies, API usage and other model data to define elaborate queries that match some bug-prone situations.

These are the different code smells which we have analysed.

- **Too big types**
  Types implementations spreading across too many lines are a burden to maintain. If you consider a reasonable limit to be say 200 lines, you can locate the types that go over that limit with the formula NbLinesOfCode > 200:

  How to fix :-
  Types with many lines of code should be split in a group of smaller types.
  1. The logic in the God Class must be splitted in smaller classes. These smaller classes can eventually become private classes nested in the original God Class, whose instances objects become composed of instances of smaller nested classes

2. Try to maintain the interface of the God Class at first and delegate calls to the new extracted classes. In the end the God Class should be a pure facade without its own logic. Then you can keep it for convenience or throw it away and start to use the new classes only
3. Unit Tests can help: write tests for each method before extracting it to ensure you don't break functionality.

- **Types with too many data members**

  Like with a large number of methods, a large number of data members can be a sign of the type having more responsibilities than it should.

  CQlink query to detect such types with a large number of data members:

  How to fix :-
  1. To refactor such type and increase code quality and maintainability, we'll have to group subsets of fields into smaller types and dispatch the logic implemented into the methods into these smaller types

- **Coupling**

  Low coupling is desirable because a change in one area of an application will require fewer changes throughout the entire application. In the long run, low coupling saves a lot of time, effort, and cost associated with modifying and adding new features to an application.
  C++ offers several tools to reduce coupling by using polymorphism. For example, abstract classes (in the sense of a class with at least one pure virtual method) or generic (template) types and methods.
  So, we can just search for abstract classes for this.

- **Types with too many methods**

  Another metric for type complexity is the number of methods. Having many methods for a type might be a sign of too many responsibilities implemented.
  CQLink query to detect this :-

Analyzing common types of malicious inputs

If an attacker provides an untrusted input to a program then this type of attack is known as injection attack. Injection attacks refer to a broad class of attack vectors. This input gets processed by an interpreter as part of a command or query. In turn, this alters the execution of that program.

Injections are amongst the oldest and most dangerous attacks aimed at web applications. They can lead to data theft, data loss, loss of data integrity, denial of service, as well as full system compromise. The primary reason for injection vulnerabilities is usually insufficient user input validation.

This attack type is considered a major problem in web security. Injection attacks, particularly SQL Injections (SQLi attacks) and Cross-site Scripting (XSS), are not only very dangerous but also widespread, especially in legacy applications.

What makes injection vulnerabilities particularly scary is that the attack surface is enormous (especially for XSS and SQL Injection vulnerabilities). Furthermore, injection attacks are a very well understood vulnerability class. This means that there are many freely available and reliable tools that allow even inexperienced attackers to abuse these vulnerabilities automatically.

Types of Injection Attacks/Malicious Input Attacks

SQL injection (SQLi) and Cross-site Scripting (XSS) are the most common injection attacks but they are not the only ones.

A) **Code Injection:**
The attacker injects application code written in the application language. This code may be used to execute operating system commands with the privileges of the user who is running the web application. In advanced cases, the attacker may exploit additional privilege escalation vulnerabilities, which may lead to full web server compromise.
Potential impact:
   - Full system compromise

B) **CRLF Injection:**
The attacker injects an unexpected CRLF (Carriage Return and Line Feed) character sequence. This sequence is used to split an HTTP response header and write arbitrary contents to the response body. This attack may be combined with Cross-site Scripting (XSS).

Potential impact:

- Cross-site Scripting (XSS)

C) **Cross-site Scripting (XSS):**
The attacker injects an arbitrary script (usually in JavaScript) into a legitimate website or web application. This script is then executed inside the victim's browser.
Potential impact:
- Account impersonation
- Defacement
- Run arbitrary JS in victim's browser

D) **Email Header Injection:**
This attack is very similar to CRLF injections. The attacker sends IMAP/SMTP commands to a mail server that is not directly available via a web application.
Potential impact:
- Spam relay
- Information disclosure

E) **Host Header Injection:**
The attacker abuses the implicit trust of the HTTP Host header to poison password-reset functionality and web caches.
Potential Impact:
- Password reset poisoning
- Cache poisoning

F) **LDAP Injection:**
The attacker injects LDAP (Lightweight Directory Access Protocol) statements to execute arbitrary LDAP commands. They can gain permissions and modify the contents of the LDAP tree.
Potential Impact:
- Authentication bypass
- Privilege escalation
- Information disclosure

G) **OS Command Injection:**
The attacker injects operating system commands with the privileges of the user who is running the web application. In advanced cases, the attacker may exploit additional privilege escalation vulnerabilities, which may lead to full system compromise.
Potential impact:
- Full system compromise

H) **SQL Injection (SQLi):**
The attacker injects SQL statements that can read or modify database data. In the case of advanced SQL Injection attacks, the attacker can use SQL commands to write arbitrary files to the server and even execute OS commands. This may lead to full system compromise.

Potential impacts:

- Authentication bypass
- Information disclosure
- Data loss
- Sensitive data theft
- Loss of data integrity
- Denial of service
- Full system compromise

I) XPath Injection:

The attacker injects data into an application to execute crafted XPath queries. They can use them to access unauthorized data and bypass authentication.

Potential impacts:

- Information disclosure
- Authentication bypass

How to detect Malicious inputs/Injection vulnerabilities?

They can be detected manually by using penetration testing but it takes a lot of time and resources. The most efficient way to detect injection vulnerabilities, which make injection attacks possible, is by using an automated web vulnerability scanner

Vulnerability Scanners

We did an analysis of vulnerability scanners and a case study of a popular vulnerability scanner.

Web Vulnerability Scanners are automated tools that scan web applications, normally from the outside, to look for security vulnerabilities such as SQL Injection, insecure server configuration,etc.There are mainly three categories of vulnerability scanners

- Network-based vulnerability scanners
  - Network vulnerability scanners scan systems across the network, by sending probes looking for open ports and services, and then probing each service further for more information, configuration weaknesses or known vulnerabilities. Network scanners are often configured either to scan internal networks, or external networks.
- Agent-based vulnerability scanners
  - Agent-based scanning is performed by installing lightweight software scanners on each device to be covered.The software scanners run local vulnerability scans and report back to a central server with the results. This type of scanners can detect a wide range of vulnerabilities, including weaknesses in software which doesn't expose ports or services for remote access.
- Web-application vulnerability scanners
  - Web application vulnerability scanners find weaknesses in web applications and websites. They work by crawling through a site or application in a similar way as a search engine would. They send a range of probes to each page to look for weaknesses.

Case Study: Acunetix

Acunetix is an automated web vulnerability scanner which scans any web application or websites that use HTTP or HTTPS protocols and are accessible through a web browser. It inspects the websites identifying vulnerabilities such as SQL injection, cross site scripting, etc.

Working of Acunetix:

Acunetix audits the web applications in two main stages

1. **Crawling** – Making use of Acunetix DeepScan, Acunetix automatically analyzes and crawls the website in order to build the site's structure. The crawling process enumerates all files, folders and inputs and is vital to ensure that all your website is scanned.
2. **Scanning** – Acunetix launches a series of web vulnerability checks against each component in your web application – in effect, emulating a hacker. The results of a scan include comprehensive details of all the vulnerabilities found within the website.

Important vulnerabilities detected by acunetix

- **Cross Site Scripting (XSS):** XSS is inserting malicious code into a victim's web application so that, when a victim browses the web application, the malicious script code is executed . When users visit a website, their browsers send HTTP requests, in which the headers include information about their browsers and operating systems. Based on this information, the users may be directed to the mobile version of website that, along with different content, may have different vulnerabilities. This has significant implications while trying to identify XSS issues. For this reason, Acunetix aims to crawl different versions of each website with different user agents.
- **SQL injection:** SQL injection vulnerability can cause the exposure of all the sensitive data of a web application database . The SQL attacker tries to insert a part of malicious SQL commands by using special variables and inserting them in to the application. The web application in turn sends these malicious commands to the target database in the server that executes them in a different purpose using legitimate query
- **Directory traversal:** Directory traversal is an attack that aims to access files and directories that are stored outside the web root folder. This vulnerability can exist in the web server or web application code. This allows the attacker to access parts of directories which are restricted, and to execute commands on the web server.