John Murray | William Tiffany

G00152680 G00\*\*\*\*\*\*

GO Online Web Application Game

isa/SWE 681\_ software assurance ASSESSMENT report



# Contents

1. INTRODUCTION
2. GO SOFTWARE GAME DESIGN
   1. DESIGN / ARCHITECTURE OF THE CODE
      1. Software Components
         1. REST API in GO
         2. Gorilla MUX
         3. GO package SQL
         4. GO package Log
         5. Resource Utilization
   2. GAME OPTIONS MENU / OPERATING INSTRUCTIONS
      1. Login / Start Game / Check Leaderboard Scores / Logout
      2. Begin New Game or Complete Previously Unfinished Game
      3. Play Competitive (1vs1) GO
      4. View Leaderboard Scores
      5. View Game Manuel (How-to-Play) Pages
   3. GO README / GAME RULES
      1. Installation Instructions / External Dependencies
      2. User/Player Game Rules
      3. Game Server Logic
3. SOFTWARE ASSURANCE ASSESSMENT / WHY WE BELIEVE IT’S SECURE
   1. Software Assurance Case Report
   2. Secure Sockets Library Imports (OpenSSL/TLS)
   3. WebApp Security Protocol Configurations
   4. Countermeasures to Common WebApp Vulnerabilities
      1. Protection Against Cross Site Scripting (XSS)
      2. Protection Against Cross Site Request Forgery (CSRF)
         1. User Input Validation
         2. Regular Expressions (Regex)
         3. Data Cleansing & Sanitation
         4. Proper Escaping of Meta-characters
      3. Protection Against SQL-Injection
   5. Protection Against Low-level Systems Attacks
      1. Safeguards to Buffer Overflow
      2. Safeguards to Format String Vulnerability
      3. Safeguards to Heap Overflow & Spraying
      4. Safeguards to Dangler Pointers (Double Frees & Use After Frees)
      5. Safeguards to Data Type Confusion
      6. Safeguards to File Path Manipulation
   6. Intrusion Detection & Prevention
      1. Logging of Unauthorized User Activity
      2. Data Encryption & Confidentiality
      3. User Privacy / Anonymity & Password Protection
      4. Ensuring Availability & Protection Against Login Failure
   7. Access Control Protection Policy
      1. User Authentication & Session Management
         1. Safeguards to Man-in-the-Middle (MITM) Attacks
         2. Safeguards to Reply Attacks
      2. Principal of Least Privilege
      3. Protection Against Malicious Administrator Backdoors
   8. Game Event and Score Logging
      1. Game Audit Trails
      2. Player Score Logging
   9. OWASP ZAP Testing
      1. Standard Mode
      2. Attack Mode
4. WRAP-UP
5. WORKS CITED

# **INTRODUCTION**

GO is one of the World’s oldest games. The ancient game of GO is at least 4000 years old and is considered to be the ultimate thinking game. Its simple rules hide enormous depth (American Go Association). Our implementation of the GO web application (WebApp) involves two players connecting to a single online game board session instance supported by a server over a secure HTTP/TLS connection with each player playing one side of the game remotely. The objective of GO is the capture the most tokens on the board thus gaining the highest percentage of points to win the game. The game ends when there are no more entries left on the board to place a token. When the game ends the player with the highest amount of points will be entered as the winner onto the virtual Leaderboard. Should either player quit the game prematurely even despite having a higher overall score the server will still recognize this as a forfeit and the player that did not quit will automatically be deemed the winner. If the game should end due to an outside unforeseen circumstance (ie. loss of Internet connectivity) whichever player maintained the highest score at the time of connection disruption will be deemed the winner.

# **DESIGN / ARCHITECTURE OF THE CODE**

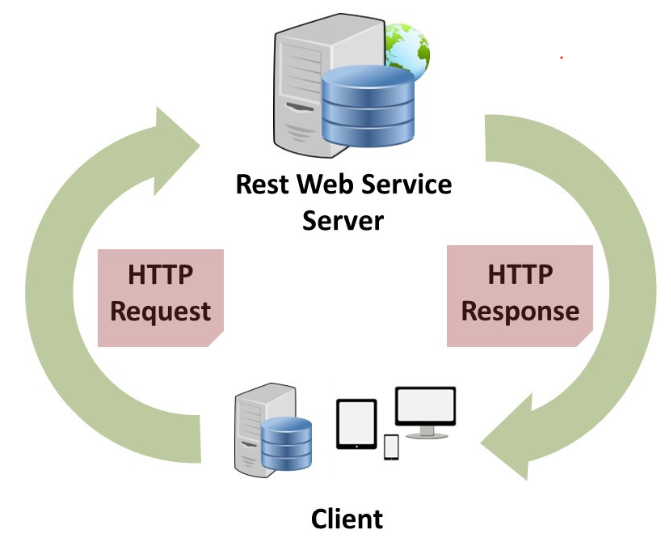
The GO online web application game was written in the The Go Language programming language developed by Google which is highly optimized toward concurrency. This chapter will outline the Software Development Lifecycle (SDLC) we employed to develop the game functionality, logic and security.

## WEB APPLICATION ARCHITECTURE

Our online version of GO employs a client server-based architecture which was modeled using the REST API framework. The server was implemented using the PostgreSQL relational database management system (RDBMS) with Gorilla Mux. Our web application servlets were implemented using the Model View Control (MVC), Front Controller and Business Delegate design patterns. For the client-side web application interface we utilized HTML Document Object Model (DOM) Selection Objects that include event bubbling, targeting and capturing. For web application deployment we utilized the REST API package manager to host the client-side command line interface onto port 8080.

## SOFTWARE COMPONENTS

The software components employed to implement the Representational State Transfer (REST) Application Programming Index (API) known as RESTful web services to provide HTTP Request and HTTP Response interoperability between the Game Client and Rest Web Service Game Server are as follows:



### REST API in GO

To build the REST API backed by PostgreSQL in The Go Language the Gorilla Mux was used for routing (Kabra, 2020). The Game Web application implements a simple REST API server which exposes endpoints allowing for the accessing and manipulating of “gameplay moves.” The operations allowable by the endpoint are:

1) Creating a new gameplay move in response to a valid HTTP POST request

2) Updating an existing gameplay move in response to a valid HTTP PUT request

3) Fetching an existing gameplay move in response to a valid HTTP GET request

4) Fetching a list of gameplay moves in response to a valid HTTP GET request

### Gorilla MUX

The Gorilla web toolkit package MUX implements a request router and dispatcher. MUX stands for “HTTP request multiplexer.” Similar to the standard http.ServeMux, mux.Router matches incoming requests against a list of registered routes and calls a handler for the route that matches the URL (The Gorilla Authors, 2012). The main features of which:

1. Gameplay move requests are matchable based on URL host, path, path prefix, schemes, header and query values & HTTP methods or using custom matchers
2. URL hosts, paths and query values can have game player names with an optional regular expression
3. Registered URLs can either be built or reversed which assists to maintaining references to stored gameplay moves
4. Implementing the http.Handler interface to ensure compatibility with the standard http.ServeMux

### GO Package SQL

The GO package Structured Query Language (SQL) includes a generic interface around other SQL-live databases. The Game Server implements the SQL package in conjunction with an SQL database driver (The Go Authors, 2009).

### GO Package Log

The GO package log implements a simple logging package which defines a type “Logger” with methods for formatting output. It also has a standard Logger accessible through helper functions Print[f|ln]. The GO logger writes to standard error and prints the date and time of each logged message. Each log message is output on a separate line. The Fatal functions call os.Exit(1) after writing the log message while the Panic functions call panic in the event of unrecoverable runtime exception errors (The Go Authors, 2009).

### Resource Utilization

The software component interface web features are accessible by all other web-based components. The Game Server includes logic to implement HTTP.ResponseWriter, HTTP.ResponseReader, HTTP.WriteHeader, HTTP.Logger as well as Fatal / Panic Error Handling.

## GAME OPTIONS MENU / OPERATING INSTRUCTIONS

## Login / Start Game / Logout

Without starting a game an online visitor is allowed to utilize and view various features of the GO web application without logging into the website. This includes viewing:

1. Rules
2. Leaderboard
3. Game Options Menu

In order to create a new account or initiate a competitive game play session an online visitor needs to first create an account and choose or otherwise be assigned a unique username in order to be added to the game registration system.

## Begin New Game or Complete Previously Unfinished Game

New online visitors have the option to either a start a new game or rejoin an existing previously saved game.

## Play Competitive (1vs1) GO

Choosing the option allows online visitors (once they have already established a unique username within the games registration system) to begin a new online game session. An empty board is populated with the option to place a game play token (or move) anywhere they wish on the board in order to initiate the first turn and start the game.

## View Leader-board Scores

The Leaderboard page allows for all visitors to the online web game to view the current running record of games won, lost or tied by individual player (unique usernames) of all previously played games.

## View Game Manuel (How-to-Play) Pages

The Official Gameplay Help Pages and list of complete rules and regulations of how to play the online Go web application game.

## GO README / GAME RULES

## Installation Instructions

Download Microsoft Visual Studio Code or open up a VI/VIM editor within Linux and save: 1) the client-side application code: igo-client.go and 2) the game Server application code: igo-server.go.

Once on the command line in order to build a brand new game project simply type at the command prompt: “go run .\igo-server.go” to run the game as a script or type: “go build .\igo-server.go” to compile and build the game as a runnable executable.

## User/Player Game Rules

Go is an ancient territorial game for 2 players each placing different types of tokens onto a symmetric game board or matrix. The objective of GO is to secure territory, but also involves the capture of enemy stones, which are added to territory gains.

Once the GO game has started players will be presented with a simple command line interface and game board matrix where player moves are token entries placed on the gameboard and entered via array indexes of both X and Y coordinates planes. Assuming the player token entry (game move) is legal the game token will be marked as an entry onto the gameboard and control will then switch to the opponent to allow for a corresponding follow-up move.

Go begins with an empty board. Player 1 enters moves first and moves then alternate between Player 1 and Player 2. When both players pass consecutively the game is over and the score is based on the resulting final board position. The player deemed the winner is the one who holds the greatest amount of territory (Go Tutoring websites).

All intersections occupied by tokens of a certain type are territory scores for the corresponding game player. Similarly, any connected empty intersections surrounded by only one token type occupying all the adjacent intersections are also scores for that game player. Connected empty intersections with adjacent stones of both player 1 & 2 token types are neutral and as such are not adding to the score of either side. The number of tokens captured by each game player is added to the running total. Whichever side has the highest score is deemed the winner.

## Game Server Logic

# **SOFTWARE ASSURANCE ASSESSMENT / WHY WE BELIEVE IT’S SECURE**

## Software Assurance Case Report

Software Assurance cases are defined to be grounds for justified confidence that a calm has been or will be achieved. According to the ***ISO/IEC 15026-2.:2011*** specifies defined structure and contents of an assurance case includes the following (Wheeler, 2019):

1. Facilitates stakeholder communications, engineering decisions (typically for safety & security claims)
2. Top-level claims for a property of a system or product
3. Arguments: systematic argumentation justifying such claim
4. Evidence & explicit assumptions underlying said argument

Our Secure Software Assurance case for the GO web application game includes various countermeasures to online security threats including implementing IPSec secure channel TLS as well as protection against common client & server-side web application vulnerability exploits and attacks.

## Secure Sockets Library Imports (OpenSSL/TLS)

OpenSSL is a secure software library for web applications used to protect data communications over computer networks against eavesdropping. OpenSSL contains an open-source implementation of the SSL and TLS protocols (OpenSSL Software Foundation, 2018).

To enhance online game security when playing over a webserver our software implements secure HTTPS Get / Post functionality for method posting calls.

## WebApp Security Protocol Configurations

Web Application online malfeasance occurs when the attacker attempts to subvert control or otherwise gain unauthorized access to unsecured files containing protected data or information. Without implementing additional web security safeguards on top of default account security or passwords leaves online accessible web server applications vulnerable to being exploited the root cause of which is improper server security configuration.

In the event of an attacker subverting control over the Go online web application our game Server is equipped to display stack tracks along with custom security logs of any unauthorized activity attempts at gaining entry into the system. This includes an error landing page to which the user will be redirected, but does not reveal pertinent data regarding why such exception was triggered.

## Countermeasures to Common WebApp Vulnerabilities

## Protection Against Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is when the attacker injects malicious code into the client web application in efforts to hack sensitive data including: cookies and browser storage. Attackers can hack sensitive data upon finding any loophole where their query reflects HTML within the web application instead of HTML entities. Attackers can inject any language that will execute within browsers without injecting server-side language.

### User Input Validation

Input validation of all untrusted inputs is vital and assists with countering many attacks (Wheeler, 2019). User Input Validation is used to mark specific action methods from game players whose game play move inputs must be validated.

The game Server is responsible for providing user input validation of the following channels (sources of input) where untrusted user data comes from the client application, including:

1. Command line arguments (setuid/setgid)
2. Environment variables (extract & erase)
3. File descriptors (stdin/stdout/stderr)
4. File names (infiles/outfiles)
5. File contents (trusted files not modified by untrusted users)
6. Web-based application inputs (HTTP POST)
7. HTTP (headers & query strings)

### Regular Expressions (Regex)

Regular expressions (Regexes) are used for filtering, checking and validating user input data. Regexes can be used to filter input, check if the data matches a certain pattern (Wheeler, 2019).

Regexes have been employed to validate the username and password files to ensure that game players create secure credentials. For each user text input a Regex is defined that describes the legal input thus making the entire pattern as limiting as possible. When the game Server receives an input a Regex library is used to verify that the pattern matches the input and will otherwise reject the input. This safeguard prevents our game Server from running such regexes provided by the attacker.

### Data Cleansing & Sanitization

Address Sanitizer (ASan) is a compilation-time countermeasure that counters buffer overflow (global/stack/heap), use-after-free & double-free and is also used to detect memory leaks while using “shadow bytes” to record memory addressability.

The Go programming language implements such data cleansing and sanitization as part of being a high-level language employing use of automatic memory and address space management.

### Proper Escaping of Metacharacters

Within POSIX extended regular expressions there exist a total of 14 metacharacters which require escaping (ie. must be preceded by a backslash “\” in order to drop their special meaning and be treated literally inside of an expression (Wikipedia, Metacharacter). Otherwise our game Server would run the risk of such unescaped code to be run within our client application program with potentially hazardous threats of untrusted scripts being run within the game.

## Protection Against Cross-Site Request Forgery (CSRF)

Cross-site Request Forger (CSRF) involves the concept of finding a loophole within a client-side API request where the attacker can send the desired data using the client web browser ID in efforts to target the API. This can occur when any updating any authenticated data or transmitting data from authenticated users like password changes.

## Protection Against SQL-Injection

SQL injection usually occurs upon asking an untrusted user for input (ie. username/userid) and instead of a name or id the untrusted user gives you an SQL statement which will unknowingly run inside your database (w3schools).

Safeguards against SQL-Injection include the use of Prepared Statements, which are features used to execute the same or similar database statements repeatedly and with high efficiency. Prepared statements are resilient against SQL injection because values that are transmitted later using a different protocol are not compiled like the statement template. If the statement template is not derived from external input, this prevents SQL injection from occurring (Wikipedia, Prepared Statement).

To protect against the attacker gaining access to all user names and passwords within our online application game we employ the use of prepared statements to counterattack any potential untrusted SQL database commands from being run within our game Server.

## Protection Against Low-level Systems Attacks

## Safeguards to Buffer Overflow

Stack-based Buffer Overflows are a form of out-of-bounds write to a memory area allocated to a buffer residing on the Stack when data written to a buffer exceeds such memory allocated for that Buffer on the Stack. Buffer Overflows can lead to program crashes, memory corruption, segmentation faults. unpredictable program behavior as well as arbitrary code execution. Buffer Overflows overwrite local variables, return addresses, function pointers and anything else that exists within higher memory (Wheeler, 2019).

After researching The Go Language we have found that as a high-level software programming language employ the follow counter measures against Buffer Overflow including use of StackGuard, special stack canary values which are checked on return to guarantee no stack return address overwrites and address space layout randomization (ASLR). ASLR randomizes the locations of key structures within memory to make reliable exploitation of vulnerabilities difficult by allocation of stack memory at random offsets.

## Safeguards to Format String Vulnerability

Format Strings consist of format parameters and data of which correspond so such parameters. Format String vulnerabilities happen when the input string is interpreted as a command by the program thus allowing the attacker to specify the format string which can lead to program crashes, reading of stack or otherwise arbitrary memory, and writing to arbitrary memory (Wheeler, 2019).

We have verified that The Go Language mitigates the format string vulnerability by disallowing format strings constructed from user input, uses secure output functions and annotation headers that enable compile-time checks for valid format strings.

## Safeguards to Heap Overflow & Spraying

Heap-based Buffer Overflows while comparable to Stack Overflows are another form of out-of-bounds write to an area of memory allocated to a buffer residing on the Heap where the buffer is dynamically allocated using malloc/new as opposed to a static Stack allocation. Heap Overflows can lead to program crashes, memory corruption and CPU or RAM denial of service. Heap overflow exploits are not as straightforward as Stack as no EIP or return address exists to provide easy control flow hi-jack several potential vulnerabilities exist including: static allocation size with user provided data that exceeds it and user provided allocation size (Greenberg, 2019).

The Go Language employs use of automatic memory management so our software code is free of any weaknesses which would otherwise exist with the manual freeing any allocated heap memory objects. Additionally, The Go Language also uses ASLR to randomize allocations of heap memory at random offsets to ensure that overwriting data allocated to the heap buffer is significantly harder.

## Safeguards to Dangling Pointers (Double Frees & Use After Frees)

***Dangling pointers (CVE-2014-1776)*** are a special breed of memory safety vulnerabilities and programming errors involving direct memory manipulation. Several vulnerabilities exist: pointer double frees, use after frees and expired pointer dereference. The consequences of which lead to “unpredictable behavior” including program crashes from reading/writing invalid memory, memory corruption from writing to valid but unintended memory location and arbitrary code execution should a dangling pointer point to a virtual function table (VFT). Modifying pointer size to include the entire process memory space allows the declared memory object to read/write all process memory thus allowing for code injection and hi-jack (Greenberg, 2019).

To circumvent such software weaknesses in our program we employed the use of a high-level software language (The Go Language) to ensure automatically explicitly set pointers and all copies of such pointers are set to NULL after freeing and call free() on any pointers pointing to out of scope objects.

## Safeguards to Data Type Confusion

***Object type confusion (CVE-2011-3521, CVE-2012-0507)*** is a particular instance when a program expects an object or resource to be a of given type that does not match the actual type. Essentially a ‘type’ is nothing but a name for a particular layout of memory hence why type confusion vulnerabilities affect how memory is read/written. Such consequences include logic errors concerning accessed objects will have different properties than expected, memory corruption of which memory allocated for a certain type with data of another type can lead to extremely volatile unpredictability (Greenberg, 2019).

We have verified that the The Go Language software programming language ensures such safeguards and support to guarantee type safety through object verification assurances on resources accessed.

## Safeguards to File Path Manipulation

File Path Manipulation occurs when PATH sets directories to search for a command line execution string whereupon the attacker has modified the path to search within different directories. Should the called program call an external command gives the attack to replace the previously trusted command string.

To protect against the aforementioned manipulation our program does not trust PATH from any untrusted sources and we have ensured that the currently directory “.” is listed after Trusted directories using the full executable name to ensure that the correct file path directory is the one being run (Wheeler, 2019).

## Intrusion Detection & Prevention

## Logging of Unauthorized User Activity

Auditing and logging are defined within Unix/Linux/POSIX systems to record system logs by storing in simple format appended specific text to a file. Loggers are configurable and programs call syslog to report something which might be logged while assigned different priority levels to each message to allow for administrators to configure and define how such logs are used (Wheeler, 2019).

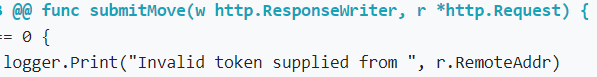
Our web application system supports secure logging per the following secure software safety messages. Logging of unauthorized user (Non-player) activity attempt at eavesdropping on current game session:





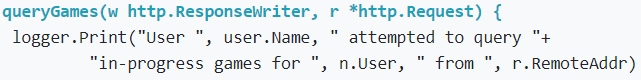
Logging of unauthorized user (Non-player) activity attempt at session hijacking and submission of illegal game move without authorization:







Logging of unauthorized user (Non-player) activity of attempted reconnaissance and querying for active in-progress game sessions:



## Data Encryption & Confidentiality

Data confidentiality is the first objective of the C.I.A. data protection security triad. Confidentiality implies the enforcement of “no unauthorized read” (Wheeler, 2019) to individuals outside the data security level or lacking the requirement permissions. To aid within the defense and safeguard of sensitive data exposure we have employed the use of AES 256 Symmetric Encryption to aid all precautionary measurements of sensitive data access control.

## User Privacy / Anonymity & Password Protection

As many user passwords are often easily guessed making them vulnerable to offline dictionary attacks, we have employed the use of storing all user chosen passwords within the game Server database as iterated salted secure SHA 256 hashes. Therefore in the event of an offline dictionary attack if our game Server’s password database is copied or stolen it will be impossible for the attacker to brute force attack, use credential stuffing or password spraying all username and password combination database entries.

## Ensuring REST Availability & Restricting HTTP methods

To ensure REST availability we have implemented the following Restrict HTTP methods within our game Server including applying a whitelist of permitted possible HTTP client application game moves for GET, POST and PUT. Additionally, we have implemented the following suggest OWASP protocols for validating untrusted REST user input:

1. Do not trust input parameter/objects
2. Validate input length, range, format & type
3. Achieve implicit validation through usage of strong types including: numbers, Booleans, dates, times or fixed data ranges within API parameters
4. Constrain string inputs using Regexes
5. Reject unexpected/illegal content
6. Make use of validation, sanitation libraries and frameworks
7. Define a request size limit and reject requests exceeding the limit with an HTTP response 413 Request Entity Too Large (OWASP)

## Access Control Protection Policy

## User Authentication & Session Management

User session management is used to facilitate secure interactions between a client application game user that has been authenticated with the hosted game Server application in response to a sequence of entered game move requests and responses associated with a unique client-side game player username entered from the client application interface.

#### Safeguards to Man-in-the-Middle (MITM) Attacks

Man-in-the-Middle (MITM) is a hidden network security exploit where the attacker secretly alters the communications between two parties where unbeknownst to the sender & receiver, they are actually communicating to each other through an invisible evil 3rd party proxy with the ability to read and modify all messages (Wikipedia, Man-in-the-middle attack). In our case the client application and the game server communicate via HTTP POST and HTTP GET requests. The attacker has the ability to intercept a relevant HTTP POST and HTTP GET game request moves submitted to the game server and sent back to the client by injecting new moves into the game.

To counter such an attack our game server has been registered a public key with a trusted certificate authority, therefore any game moves submitted to the server by the client application are encrypted using the game servers public key with ensures both game move validation and client application move non-repudiation.

#### Safeguards to Reply Attacks

Replay attack is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated at a later time (Wikipedia, Replay Attack). An instance of such attack would be when the attacker captures the game move from the client on the way to the Server and attempts to replay the game move at a later time to the game Server.

To patch the replay attack vulnerability, we have incorporated the use of a Nonce (ie. number used once, freshness token, or timestamp) using web cookies. The Client application requests a game move with its own Nonce, the game Server will then challenge the Client with its own Nonce such that the Client can encrypt both Nonces inside a game move request and forward to the game Server, if the Server can decrypt its own Nonce the Client application has been authenticated. Now should an attacker attempt to reply any client game moves to the game Server the Nonce timestamp would have already expired and the game move would simply get dropped.

#### Safeguards to Session Hijacking

Session hijacking is a web application vulnerability involving the exploitation of a web session where the attacker will attempt to hack the session ID’s or user data from the client. Once succeeded in hijacking client the attacker will set the session data in the browser within the same server hosted website and when the attacker refreshes the browser, they will automatically be logged into the same website session (Agira Technologies).

## Principal of Least Privilege

Principal of Least Privilege ensures that each game player operates using the fewest privileges possible thus limiting the number of potential interactions among privilege processes (including game logging, player score keeping and number of games won.)

To enforce the Principal of Least Privilege our program ensures complete mediation and non-bypassability such that every player access attempt is checked and cannot be subverted. Our program minimizes privileges granted by not granting root and avoids creating setuid root programs. Our game server does all the access checking as our client-side application is untrusted. Privileges have also been separated such that multiple checks are initiated upon game player moves which are dependent on more than one condition. Our web application game maintains a separate administrative interface for the server with different privileges with the client-side being granted the fewest amount of privileges possible.

## Protection Against Malicious Administrator Back-doors

A backdoor refers to any method by which authorized and unauthorized users are able to circumvent normal security measures and gain high-level access (ie. root access) on a software application (Malwarebytes).

In order to prevent our online game application against harmful built-in backdoors we have employed the following countermeasures: changing all default user passwords, choosing the applications and secure libraries carefully and implementing the game Server as a cybersecurity solution.

## Game Event & Score Logging

## Game Audit Trails

## Player Score Logging

## OWASP ZAP Testing

## Standard Mode

## Attack Mode

# **Wrap-Up**

The GO online web application game was designed to teach us how to successfully implement the essential fundamentals of secure software design and also how to write a quality software assurance case report. Throughout the development of the web application game we taught ourselves the basic core functionality of writing secure software programs. Proper use of OpenSSL web certificates and importing TLS network security libraries hosted over a secure socket connection to defend data confidentiality, integrity and user input validation to protect against arbitrary code execution found in common server-side application and client-side web browser attacks, and the logging of surreptitious unauthorized user activity are a few of the many highlights which we learned to deploy within the interface of the secure online web game application.

# **Works Cited**

Agira Technologies. (n.d.). Explaining XSS, CSRF And Session Hijacking. In *agiratech.com/xss-csrf-and-session-hijacking.*

American Go Association. (n.d.). www.usgo.org.

Go Tutoring websites. (n.d.). *www.playgo.to/index-e.html*.

Greenberg, B. (2019). *ISA 564 Security Laboratory Course Slides.* Fairfax: George Mason University.

Kabra, K. (2020). Building and Testing a REST API in GO with Gorilla Mux & PostgreSQL. *Semaphore*.

OpenSSL Software Foundation. (2018). OpenSSL/TLS Tookit. *openssl.org/support*.

OWASP. (n.d.). Restrict HTTP Methods. In *cheatsheetseries.owasp.org/cheatsheets/REST\_Security\_Cheat\_Sheet.html.*

The Go Authors. (2009). Go Programming Language Documentation. *golang.org/LICENSE*.

The Gorilla Authors. (2012). Gorilla web toolkit. *gorillatoolkit.org*.

w3schools. (n.d.). SQL injection. In *w3schools.com/sql/sql\_injection.asp.*

Wheeler, D. D. (2019). *SWE 681 / ISA 681 Secure Software Design & Programming Course Slide.* Fairfax: George Mason University.

Wikipedia. (n.d.). Man-in-the-middle attack. In *en.wikipedia.org/wiki/Man-in-the-middle\_attack.*

Wikipedia. (n.d.). Metacharacter. In *en.wikipedia.org/wiki/Metacharacter.*

Wikipedia. (n.d.). Prepared Statement. In *en.wikipedia.org/wiki/Prepared\_statement.*

Wikipedia. (n.d.). Replay Attack. In en.wikipedia.org/wiki/Relay\_attack.