International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

A Comprehensive Approach To Cyber Threats: Analysis Of Suspicious Profiles And Protection Of Web Applications

Kabykenova Sabina[1], Kanatov Nurzhas[1], Maksim Yavich[2]

[1]Al-Farabi Kazakh National University,Almaty,Kazakhstan
[2]Caucasus University,Tbilisi,Georgia

A B S T R A C T :

This study is devoted to a comprehensive approach to cyber threats that combines web application protection and suspicious profile analysis. The main focus is on preventing SQL injections and XSS attacks using Go middleware. The implemented mechanism filters malicious requests, blocking attempts to inject dangerous code. Moreover, user analysis is integrated into the system, allowing it to identify potentially malicious profiles. To achieve this, the system records IP addresses, timestamps of user activity, and distinctive behavioral patterns indicating automated attacks. As a result, it becomes possible not only to block individual malicious requests but also to identify suspicious users who pose a security threat. The proposed methods were tested using automated HTTP requests containing SQL injections, malicious JavaScript code, and attempts to bypass CSRF protection. The experiment demonstrated that the approach effectively prevents attacks and detects abnormal activity. Consequently, the proposed technique enhances web application security and contributes to the early detection of cyber threats. Despite these results, further research involves integrating more advanced machine learning mechanisms for anomaly detection, as well as the use of prepared SQL queries and built-in XSS filters to strengthen protection.

Keywords: cybersecurity, fake accounts, machine learning, social engineering, multi-factor authentication, SQL injections.

1. Introduction

Modern cyber threats are becoming increasingly sophisticated, requiring a multi-layered approach to security. Among the most pressing threats are fake profiles on social media and attacks targeting web applications. Fake accounts can be used to spread disinformation, conduct social engineering, commit fraud, and even launch attacks against businesses. Malicious actors often create fake profiles, impersonating well-known individuals or organizations to gain users' trust and access confidential information. Information shared on social platforms is not originally intended for unauthorized use. However, there is a risk that this data may be exploited by attackers to incite conflicts, generate video content that undermines the dignity, reputation, or honor of users, or manipulate public opinion. Fake accounts also pose a threat to national security. Video content and personal data from social networks can be accessed by employers, relatives, debt collectors, cybercriminals, and others.

At the same time, web applications are increasingly targeted by various types of attacks, such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). These vulnerabilities can lead to data breaches and service disruptions. The goal of web security is to prevent these and other types of cyberattacks. More precisely, web protection refers to methods of ensuring the security of online resources from unauthorized access, use, modification, destruction, or disruption. Effective website protection requires close attention to its development, including the structure of the application, the design of update and recovery policies, and the implementation of client-side encryption. A reliable defense must involve a comprehensive approach—combining the analysis of suspicious profiles with the use of modern cybersecurity tools.

2. Research Methodology

In the context of growing cyber threats, the modern information environment requires a comprehensive approach to protecting data and users. This study is aimed at analyzing suspicious profiles within digital ecosystems and protecting web applications from common attacks. The methodology covers two key areas: detecting anomalous activity on social networks and developing effective protection mechanisms for web systems using the Go programming language.

Identifying fake accounts and automated bots is a critical task in ensuring information security. The following methods are used to detect fake profiles:

- Content analysis – examining texts and media files for anomalies, identifying template-based or automatically generated messages.
- Socio-cultural analysis – studying profile activity: frequency of posts, interaction patterns, content repetition, and temporal regularities.
- Machine learning methods – applying classification and clustering models to detect suspicious behavior patterns.
- Network structure analysis – identifying links between accounts, detecting artificially created communities and automated bot networks.

These methods help minimize the impact of disinformation, fraud, and social manipulation in digital environments.

In parallel with the analysis of threats posed by suspicious profiles, the study focuses on developing mechanisms to protect web applications from widespread attacks such as SQL injection, XSS attacks, and CSRF. The following methods are applied:

- Input filtering and validation – preventing the injection of malicious code into user requests through parameterized SQL queries and escaping.
- Use of secure HTTP headers – configuring Content Security Policy (CSP), HTTP Strict Transport Security (HSTS), and X-Frame-Options to guard against cross-site attacks.
- Secure authentication and session management – password encryption, use of JWT and OAuth 2.0 to protect user data.
- Implementation of CSRF tokens – preventing request forgery using unique one-time tokens.
- Logging and monitoring – analyzing traffic and events to detect attacks, integrating with SIEM systems for early threat detection.

The use of the Go programming language for implementing security mechanisms is driven by its secure architecture, built-in memory management tools, and high performance.

Combining suspicious profile analysis methods with web application protection practices enables the construction of an effective cyberattack mitigation system. This approach not only safeguards user data but also enhances overall digital security, which is crucial in an environment of constantly evolving cyber threats.

2.1 Web Application Protection

Ensuring the security of web applications requires the implementation of multi-layered defense measures. One of the key tools in this process is a Web Application Firewall (WAF), which filters incoming and outgoing traffic to protect applications from attacks such as SQL injections, XSS, and CSRF. A WAF analyzes requests for malicious patterns and blocks suspicious activity, preventing the exploitation of vulnerabilities.

In addition, it is recommended to implement Multi-Factor Authentication (MFA) to enhance the security of user accounts. MFA requires users to provide multiple forms of identity verification, which significantly complicates unauthorized access attempts by attackers.

Regular software updates and patch management help eliminate known vulnerabilities and reduce the risk of successful attacks. It is also crucial to conduct regular penetration testing to identify and address potential vulnerabilities before they can be exploited by malicious actors.

Finally, educating developers and staff on secure coding practices and security best practices contributes to building more secure applications and reduces the likelihood of human error leading to vulnerabilities.

Web application protection is a critically important aspect of developing and operating modern online services. With the growing number and complexity of cyberattacks, a comprehensive security approach is essential—one that incorporates various methods and tools. Web applications are exposed to a wide range of threats, the most common of which include:

1. SQL Injection: An attack in which a malicious actor injects harmful SQL code into database queries, potentially leading to unauthorized access or data modification.
2. Cross-Site Scripting (XSS): A vulnerability that allows malicious scripts to be embedded in web pages, which can result in data theft or unauthorized actions performed on behalf of users.
3. Denial-of-Service (DDoS) Attacks: Attacks aimed at overloading a server with excessive requests, rendering the service unavailable to legitimate users.

Methods of Web Application Protection

1. Using a Web Application Firewall (WAF): A WAF analyzes incoming and outgoing traffic of a web application, automatically detecting and blocking application-layer attacks. It helps defend against a broad range of threats, including SQL injections and XSS attacks.
2. Implementing Secure Development Practices: Following secure coding principles—such as input validation and escaping, using prepared statements for database operations, and regularly updating dependencies—reduces the risk of vulnerabilities.
3. Conducting Regular Security Testing: Frequent security audits, including penetration testing and vulnerability assessments, help identify and eliminate weaknesses before attackers can exploit them.
4. Staff Training: Educating developers and administrators on the fundamentals of information security increases awareness of potential threats and methods to prevent them.
5. Monitoring and Logging: Setting up monitoring and logging systems enables timely detection of suspicious activity and helps respond to security incidents effectively.

3. Implementing Web Application Protection in Golang as a Solution to SQL Injection, XSS, and CSRF Threats

We chose the Go programming language because it offers high performance, built-in security, and convenient tools for developing web applications. Its strong typing and memory safety features help prevent vulnerabilities, while the standard libraries enable efficient implementation of malicious data filtering, attack protection, and event logging. Thanks to its simplicity and support for concurrent middleware handlers, Go is an excellent choice for building secure web applications protected against SQL injection, XSS, and CSRF attacks. The first step in protection involves creating regular expressions to detect SQL injections and XSS attacks. Figure 1 shows how input request parameters are analyzed to identify potentially harmful patterns.

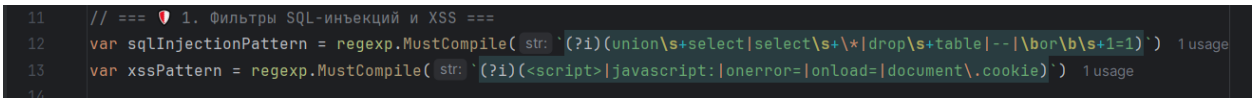

Figure 1.
Regular expressions for filtering attacks

Figure 2 shows the process of filtering SQL injections and XSS attacks in middleware. This middleware checks the parameters of incoming HTTP requests using regular expressions. If SQL injection or XSS code is detected in the parameters, the request is blocked with an error message.

```
// === 🔍 Фильтр SQL-инъекций и XSS ===
func securityMiddleware(next http.Handler) http.Handler {  1 usage
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        for key, values := range r.URL.Query() {
            for _, value := range values {
                if sqlInjectionPattern.MatchString(value) {
                    logAttack(r,  attackType: "SQL Injection", key, value)
                    http.Error(w,  error: "❌ SQL-инъекции запрещены!", http.StatusForbidden)
                    return
                }
                if xssPattern.MatchString(value) {
                    logAttack(r,  attackType: "XSS Attack", key, value)
                    http.Error(w,  error: "❌ XSS запрещен!", http.StatusForbidden)
                    return
                }
            }
        }

        next.ServeHTTP(w, r)
    })
}
```

Figure 2.
Filtering SQL Injections and XSS in Middleware

```
// === 🔍 CSRF-защита (проверка CSRF-токена) ===
func csrfMiddleware(next http.Handler) http.Handler {  1 usage
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        if r.Method == "POST" {
            token := r.Header.Get( key: "X-CSRF-Token")
            if token == "" || token != "secure-token" {
                logAttack(r,  attackType: "CSRF Attack",  param: "",  value: "")
                http.Error(w,  error: "❌ CSRF-защита сработала!", http.StatusForbidden)
                return
            }
        }

        next.ServeHTTP(w, r)
    })
}
```

Figure 3.
CSRF Token Validation Middleware

Figure 3 shows the middleware that checks for the presence and validity of the X-CSRF-Token header in POST requests. If the header is missing or its value does not match the expected token, the request is rejected. At the final stage, an HTTP server is created that uses the middleware for protection and runs on port 8080, as shown in Figure 4.

```go
// === 🌎 Обработчик запросов ===
func homeHandler(w http.ResponseWriter, r *http.Request) { 1 usage
    fmt.Fprintf(w, format: "✅ Приложение работает. Ваш ввод: %s", html.EscapeString(r.URL.Query().Get( key: "input")))
}

// === 🚀 Запуск сервера ===
func main() {
    mux := http.NewServeMux()
    mux.Handle("/", securityMiddleware(csrfMiddleware(http.HandlerFunc(homeHandler))))

    log.Println( v...: "🚀 Сервер запущен на :8080")
    http.ListenAndServe( addr: ":8080", mux)
}
```

Figure 4.
Launching the Web Server with Attack Protection

This code implements protection for a web application against the most common attacks: SQL injection, XSS, and CSRF. The use of middleware allows centralized request handling and blocking of malicious data.

Testing of the implemented protection layer is performed using simple requests.

The curl command is used to send requests to the specified address. In one test, a request containing an SQL injection with the keywords UNION SELECT is sent. The server is expected to reject the request and return an error message, as shown in Figure 5.

```
PS C:\Users\Нуржас\Desktop\topic> curl "http://localhost:8080/?input=UNION SELECT * FROM users"
curl : ✕SQL-инъекции запрещены!
строка:1 знак:1
+ curl "http://localhost:8080/?input=UNION SELECT * FROM users"
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
```

Figure 5.
Sending an SQL injection request and application response

When testing protection against XSS, a malicious JavaScript code is sent within a request parameter. As shown in Figure 6, the server is expected to block the request and respond accordingly.

```
PS C:\Users\Нуржас\Desktop\topic> curl "http://localhost:8080/?input=<script>alert('XSS')</script>"
curl : ✕XSS запрещен!
строка:1 знак:1
+ curl "http://localhost:8080/?input=<script>alert('XSS')</script>"
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
```

Figure 6.
Sending a request with malicious JS code and application response

Next, testing the protection against CSRF attacks: a POST request is sent without a CSRF token, and the server rejects it, returning an error message, as shown in Figure 7.

```
PS C:\Users\Нуржас\Desktop\topic> $headers = @{"X-CSRF-Token" = "secure-token"}
PS C:\Users\Нуржас\Desktop\topic> Invoke-WebRequest -Uri "http://localhost:8080/" -Method Post
Invoke-WebRequest : ✕CSRF-защита сработала!
строка:1 знак:1
+ Invoke-WebRequest -Uri "http://localhost:8080/" -Method Post
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
```

Figure 7.
Sending a POST request without a CSRF token and application response

The developed server written in Golang successfully implements protection for the web application against common attacks such as SQL injection, XSS (cross-site scripting), and CSRF (cross-site request forgery). Testing showed that:

- The XSS filter blocks the input of dangerous tags like <script> and other constructs capable of executing malicious JavaScript code.
- SQL injections are prevented using regular expressions that block typical SQL queries such as UNION SELECT, OR 1=1, and DROP TABLE.
- CSRF protection requires the mandatory presence of the X-CSRF-Token header when sending POST requests, which prevents unauthorized actions performed on behalf of the user.

Thus, the implemented middleware functions effectively protect the application at an initial level. However, in real-world scenarios, it is recommended

to use more comprehensive protection mechanisms, such as prepared SQL statements, built-in XSS filters provided by frameworks, and standard CSRF solutions (for example, server-side token generation and validation).

4. Result and Discussion

The application of the described methods significantly enhances the level of cybersecurity. Detecting and removing fake accounts helps reduce the spread of misinformation and lowers risks associated with social engineering. Protecting web applications using WAF and other measures prevents exploitation of vulnerabilities and safeguards user data from compromise. Regular updates and penetration tests enable timely identification and remediation of securitн flaws, maintaining a high level of protection. Implementing multi-factor authentication reduces the likelihood of unauthorized account access, even if passwords are compromised. Training employees in cybersecurity principles raises overall awareness and fosters a security culture within the organization. A comprehensive cybersecurity approach that combines technical measures, processes, and training provides the most effective defense against modern threats. In 2024, an independent test of Russian Web Application Firewall (WAF) systems demonstrated the effectiveness of these solutions in protecting web applications from various attacks. According to Positive Technologies, in 2017, the average number of critically dangerous vulnerabilities per web application was two, highlighting the need for regular updates and penetration testing to maintain a high level of security. In 2022, the global WAF market was valued at 5.43 billion USD and is forecasted to reach 19.75 billion USD by 2030, indicating the growing recognition of the importance of web application protection. In 2024, Meta faced criticism for slow removal of fraudulent advertisements, resulting in significant financial losses for users. This underscores the necessity of effective detection and removal of fake accounts to mitigate risks related to social engineering.

5. Conclusion

The implementation of multi-factor authentication and employee cybersecurity training remain key measures for preventing unauthorized access and fostering a security-conscious culture within organizations. The conducted study examined a comprehensive approach to securing web applications written in Golang, as well as methods for protecting against common attacks such as SQL injection, XSS, and CSRF. The analysis of modern cyber threats revealed that insufficient protection of web applications can lead to user data compromise, service disruption, and financial losses. Golang offers built-in mechanisms for securing web applications, including strong typing, safe input handling, and support for modern authentication and authorization methods. This work reviewed key attack prevention techniques: using ORM libraries to protect against SQL injections, escaping input data to minimize XSS threats, and implementing tokens and security policies to prevent CSRF attacks. The research results confirm that combining various protection methods significantly improves the security level of web applications. However, given the ongoing evolution of cyber threats, regular updates of security mechanisms, monitoring of new attack vectors, and adaptation of approaches are necessary. Further research in this field may focus on integrating machine learning methods for anomaly detection, developing automated protection systems, and creating comprehensive solutions that ensure web application security at all levels.

REFERENCES

[1] A. AlZubi, J. Alqatawna, M. Abu-Sharkh, and H. Faris, "Detecting Fake Accounts on Social Media Using Supervised Machine Learning Algorithms," Journal of Information Science, vol. 48, no. 1, pp. 34-51, 2022.
[2] S. Cresci, F. Lillo, D. Regoli, S. Tardelli, and M. Tesconi, "$FakeFollowers: Analyzing and Detecting Fake Users in Social Media," ACM Transactions on the Web, vol. 14, no. 3, pp. 1-28, 2020.
[3] A. El Azzaoui, M. Benatia, and A. Oubani, "Fake Accounts Detection in Online Social Networks Using Graph-Based Approaches," Expert Systems with Applications, vol. 216, p. 119417, 2023.
[4] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini, "The Rise of Social Bots," Communications of the ACM, vol. 59, no. 7, pp. 96-104, 2016.
[5] S. Kumar and N. Shah, "False Information on Web and Social Media: A Survey," in Proceedings of the 25th International Conference on World Wide Web (WWW '16 Companion), 2016, pp. 1-6.
[6] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, "Fake News Detection on Social Media: A Data Mining Perspective," ACM SIGKDD Explorations Newsletter, vol. 19, no. 1, pp. 22-36, 2017.
[7] S. Kaur, S. Kumar, S. Venkatramanan, L. Ramaswamy, and M. Marathe, "Detecting Malicious Users in Online Social Networks with Graph Neural Networks," IEEE Transactions on Knowledge and Data Engineering, vol. 34, no. 9, pp. 4512-4526, 2022.
[8] K. C. Yang, O. Varol, C. A. Davis, E. Ferrara, A. Flammini, and F. Menczer, "Arming the Public with AI to Counter Social Bots," Human Behavior and Emerging Technologies, vol. 1, no. 1, pp. 48-61, 2019.
[9] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu, "The Socialbot Network: When Bots Socialize for Fame and Money," in Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC), 2011, pp. 93-102.
[10] E. Alothali, N. Zaki, E. A. Mohamed, and H. Alashwal, "Detecting Social Bots on Twitter: A Literature Review," in Proceedings of the International Conference on Big Data Analytics and Knowledge Discovery (DaWaK), 2018, pp. 41-55.
[11] T. Hwang, S. Kim, and Y. Kim, "Fake Profile Detection in Online Social Networks Using Behavioral Features and Machine Learning," Information Sciences, vol. 629, p. 119862, 2023.
[12] N. Chavoshi, H. Hamooni, and A. Mueen, "Identifying Correlated Bots in Twitter," Social Network Analysis and Mining, vol. 7, no. 1, p. 11, 2017.

[13] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao, "Detecting and Characterizing Social Spam Campaigns," in Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10), 2010, pp. 35-47.
[14] O. Varol, E. Ferrara, C. Davis, F. Menczer, and A. Flammini, "Online Human-Bot Interactions: Detection, Estimation, and Characterization," in Proceedings of the 11th International AAAI Conference on Web and Social Media (ICWSM), 2017, pp. 280-289.
[15] H. Singh, S. Sharma, S. Sehgal, and P. Gupta, "A Review on Detection and Analysis of Fake Profiles in Social Networks," International Journal of Advanced Computer Science and Applications, vol. 13, no. 1, pp. 245-257, 2022.
[16] A. S. Abrahams, J. Jiao, G. A. Wang, W. Fan, and Z. Zhang, "Social Media Fake Account Detection by 3-Way Decision-Based Deep Learning," Decision Support Systems, vol. 140, p. 113429, 2021.
[17] H. Q. Tran, H. Kavak, and V. S. Subrahmanian, "Detecting Fake Twitter Users Using Node Embeddings and Supervised Learning," IEEE Transactions on Computational Social Systems, vol. 7, no. 3, pp. 755-768, 2020.
[18] C. F. Hsu, C. C. Hsu, T. Y. Lin, H. Y. Wang, and C. Y. Lin, "Detecting Fake Profiles on Social Media Using Self-Supervised Learning," Expert Systems with Applications, vol. 221, p. 119814, 2023.