

Article

BlackWatch: Increasing Attack Awareness within Web Applications

Calum C. Hall ¹, Lynsay A. Shepherd ^{2,*}  and Natalie Coull ²

¹ MWR InfoSecurity, London SE1 3RS, UK; calumhall96@gmail.com

² School of Design and Informatics, Abertay University, Dundee DD1 1HG, UK; n.coull@abertay.ac.uk

* Correspondence: lynsay.shepherd@abertay.ac.uk; Tel.: +44-01382-308685

Received: 15 January 2019; Accepted: 11 February 2019; Published: 15 February 2019



Abstract: Web applications are relied upon by many for the services they provide. It is essential that applications implement appropriate security measures to prevent security incidents. Currently, web applications focus resources towards the preventative side of security. While prevention is an essential part of the security process, developers must also implement a level of attack awareness into their web applications. Being able to detect when an attack is occurring provides applications with the ability to execute responses against malicious users in an attempt to slow down or deter their attacks. This research seeks to improve web application security by identifying malicious behavior from within the context of web applications using our tool BlackWatch. The tool is a Python-based application which analyzes suspicious events occurring within client web applications, with the objective of identifying malicious patterns of behavior. This approach avoids issues typically encountered with traditional web application firewalls. Based on the results from a preliminary study, BlackWatch was effective at detecting attacks from both authenticated and unauthenticated users. Furthermore, user tests with developers indicated BlackWatch was user-friendly, and was easy to integrate into existing applications. Future work seeks to develop the BlackWatch solution further for public release.

Keywords: web application firewall; intrusion prevention; software security; web application security; attack awareness; cyber security

1. Introduction

Society is becoming increasingly dependent on the online services provided by many organizations [1,2]. Web applications enable users to interact with organizations using their own portable devices. However, with the continued growth of online services comes the burden of ensuring appropriate security is implemented. Web applications are often the most public facing component within many organizations, and are therefore the typical attack vector chosen by malicious attackers [3]. In Verizon's 2018 Data Breach Incident Report, it was stated that approximately 19.5% of data breaches analyzed by the Verizon research team were a result of web application attacks [4].

Developers often focus their resources towards the preventative side of application security, to reduce incidents of web application attacks. However, work presented in this paper aims to demonstrate that prevention methods alone are not enough to protect an application—it is essential for developers to implement appropriate methods to detect that attacks are occurring against their applications in real time. When targeting an application, very rarely will an attacker discover a vulnerability on their first attempt. Thus, if applications can accurately identify the behavior of malicious users in a timely manner, then it becomes possible to execute the necessary responses required to deter or slow down these application attacks. This in turn protects the web application itself, and protects innocent users from engaging with a vulnerable web application.

Following an overview of existing methods used for web application security, we present our solution, BlackWatch (named after the infantry battalion of the Royal Regiment of Scotland). The paper contains the following contributions:

1. A discussion highlighting why preventative methods are insufficient to protect web applications.
2. An overview regarding issues with existing Web Application Firewalls.
3. BlackWatch, an extensible security solution which identifies malicious behavior from within the context of a web application, and increases attack awareness.

When creating an application, developers have a clear understanding of the legitimate usage that is expected at each stage, and hence they are able to accurately identify when a user's actions are 'unexpected' or 'suspicious'. An example scenario involves a web application that allows users to upload a profile picture. The application would most likely limit the type of files users can upload through code similar to that shown in Figure 1.

```

if (uploadedFile == 'jpg' OR uploadedFile == 'png') then
  | accept user file;
else
  | return "invalid file type";
end

```

Figure 1. File Upload.

This preexisting function could be easily altered by a developer to implement further checks. For example, the application could check to see if the uploaded file is a PHP file, and if so it is highly likely that this is an attacker testing for a 'malicious file upload' vulnerability (Figure 2).

```

if (uploadedFile == 'jpg' OR uploadedFile == 'png') then
  | accept user file;
else
  | if (uploadedFile == 'php') then
  | | report the users malicious activity;
  | end
  | return "invalid file type";
end

```

Figure 2. File Upload Check.

The BlackWatch solution aims to monitor this type of suspicious behavior, with the objective of analyzing user events to determine if their actions are indicative of malicious intent.

This paper seeks to discuss the current security measures available for web applications in relation to firewalls in Section 2. The design and development of the BlackWatch solution will be covered in Section 3, before evaluating the tool and presenting the results in Section 4. Finally, the results will be discussed in Section 5, and conclusions are drawn in Section 6.

2. Background

Web applications are a highly targeted component of many organizations therefore it is essential that the security industry is actively improving methods used to defend against attacks. Insecure web applications are detrimental to online businesses and can place end-users at risk of interacting with vulnerable sites [5]. The section demonstrates the current state of web application development, highlighting how alternative methods may improve attack awareness.

2.1. Attack Awareness

In recent years, the security industry has switched focus from attack prevention to attack detection. Attack awareness has been built into several applications, such as intrusion detection systems [6]. While prevention is a vital aspect of security, detection must be taken into consideration. The switch in focus has been discussed by industry experts: to ensure an organization has a strong level of security in place, they must move forward with the assumption that their prevention methods will be bypassed [7]. This approach allows organizations to focus their efforts on being security aware, to the extent where they can detect attacks in a timely manner, therefore allowing them to respond effectively. Recently it has been stated that “*security focus has shifted from prevention towards resilience and a broader set of capabilities including timely detection and the ability to respond to live incidents*” [8].

2.2. Current Industry Methods

Within industry a variety of methods and tools are recommended for detecting and preventing attacks against web applications. The Payment Card Industry Data Security Standards suggests two alternative solutions, one of which must be implemented to ensure compliance regulations are met [9]:

- Regular source code and component reviews
- Detection and prevention of web-based attacks via the use of an automated solution, i.e., a web application firewall

Due to the extensive resources required for source code and component reviews, most organizations choose to implement a web application firewall (WAF) instead. WAF solutions can effectively protect web applications from many attacks; however, to ensure effectiveness, organizations must invest a large quantity of resources into configuring and maintaining the solution. High-Tech Bridge [10] identify the issue by stating “*ModSecurity is a very powerful and flexible WAF; however, it requires a lot of time and effort to avoid false negatives and prevent false positives*”. While High-Tech Bridge [10] are very positive about the usage of WAFs, not all security researchers express the same opinion. Kolochenko published an article exploring the five main downfalls facing WAFs [11]:

- “*Negligent deployment, lack of skills and different risk mitigation priorities*”—There is often a lack of accountability within organizations as to the entity responsible for maintaining the WAF.
- “*Deployment only for compliance purposes*”—Due to the PCI-DSS compliance regulations, WAF solutions are often deployed without ever being properly configured or managed.
- “*Complicated diversity of constantly evolving web applications*”—Web applications are constantly evolving and implementing new features, therefore keeping WAF rules up-to-date can be a challenging and time-consuming task.
- “*Business priorities domination over cyber security*”—A key concern of any security solution is the risk of affecting legitimate application usage. To prevent false positives from affecting user, organizations switch WAFs to ‘detect only’ mode. This means no preventative action is taken. Further to this, many WAFs feature poor reporting capabilities: unless an employee is tasked with manually checking the reported incidents, these attacks often go unnoticed.
- “*Inability to protect against advanced web attacks*”—WAFs often work by blocking known attacks and identifying events that are predetermined as malicious; therefore new, advanced attacks will not be detected.

It is apparent that the most common downfall of WAF solutions is that often they are not configured properly, and default configurations are not effective. Security researcher Mazin Ahmed [12] published work into WAFs where he demonstrated why default configurations are not effective. The approach of ‘blacklisting’ that many WAFs use consists of defining a set of data that represents known-bad attacks [13]. The issue with this type of approach as demonstrated by Ahmed [12] is that this approach to security can be easily bypassed. Within this research, eight well renowned solutions

were tested using their default configurations, and by using a variety of different methods Ahmed managed to bypass all eight solutions.

2.3. Implementing Security into the Software Development Lifecycle (SDLC)

One of the most difficult realities faced within the security industry, is that for an application to be secure, developers must consider every attack vector. However, for an application to be considered ‘insecure’ an attacker only needs to find one area that has not been fully secured. To combat this, there has been a concerted effort to encourage developers to include security into the software development life cycle (SDLC).

This issue was acknowledged by Jones and Rastogi [14] in 2004, who stated “*The reality is that information security is an afterthought for many organizations*”. Goertzel [15] also considered the problem, arguing that enhancing the SDLC would produce secure software. Similar thoughts have also been voiced by Jerry Hoff—Vice President of the static code analysis division at WhiteHat Security in an article on DARKReading, stating the need to secure the SDLC [16]. Hoff identifies that every organization will have their own approach to the SDLC and hence there is no one method on how to implement security throughout the process. However, he states that what is important is that once an organization has defined the security approach that work best for them, it is essential that they are consistent throughout development.

There are a variety of sources available providing guidance for developers implementing security into their SDLC, for example, SANS produced a document containing a variety of high-level suggestions on how to implement security at some of the key stages of the SDLC [17]. Hasan et al. [18] have also stressed the importance of integrating the SDLC. A model for ensuring security is built into the SDLC has previously been proposed [19], and others have investigated the use of such models, subsequently publishing a case study [20].

The objective of this approach is not to replace the need for alternative security testing—such as security audits and source code reviews, but rather to encourage developers to think about the security implications involved with the components they implement.

Furthermore, security researcher Peter Gregory stated that “*Organizations that fail to involve information security in the life cycle will pay the price in the form of costly and disruptive events.*” [21]. With developers now beginning to have an appreciation for the need for secure programming practices, it is possible that attack detection methods could be implemented into the application itself.

2.4. AppSensor

Although several commercial and open source WAFs are available [22,23], and self-healing WAFs have been explored [24], there are alternative methods of enhancing web applications thus increasing attack awareness. This research focuses on the approach of identifying malicious behavior from within the context of a web application.

In 2008 the Open Web Application Security Project (OWASP) began working on a project called AppSensor [25]. AppSensor is an open source project originally developed by Michael Coates. The project was developed with the concept of being able to identify attacks from within the context of an application. If a developer is aware of the legitimate usage of their application, then they can confidently identify illegitimate and unintended usage. In 2017, OWASP stated that “*The future of application defense is a system that can understand custom attacks against an application, correlate them against a malicious attacker, and react in real time to contain and eliminate the threat*” [25]. While the AppSensor project provides its own solution that can be adopted within a security environment, the main focus of this project is more to broadcast the idea of this type of attack detection. It has been stated that “*the idea behind OWASP AppSensor project is that it is not a product, like a WAF would be, but rather an idea with a reference implementation in a Java framework*” [26].

The objective of Thomassen’s work was to provide a comparison of the detection capability offered by WAFs, intrusion detection systems, and AppSensor. Throughout the investigation,

a variety of attacks based on the top ten most common web application vulnerabilities [27] were used to determine the effectiveness of each solution. The results demonstrated that the WAF used—ModSecurity—provided satisfactory attack detection coverage; however, AppSensor proved a greater rate of attack detection.

With the concern for security growing throughout the cyber community, it was apparent there was a need for new security solutions and approaches to be introduced. In 2017 OWASP released an up-to-date list of the most common web application vulnerabilities, number ten on that list being insufficient logging and monitoring—“*exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident*” [28]. With this issue being widely recognized within the industry, web application developers need to adopt more proactive methods for identifying attacks against their web applications.

BlackWatch seeks to address this issue, exploring how to improve web application security by identifying malicious behavior from within the context of a web application. The following section will detail the methodology behind creating and evaluating the solution.

3. Methodology

3.1. AppSensor Investigation

When considering the identification of malicious behavior from within the context of a web application, OWASP’s AppSensor was an influential solution in the field. Prior to the development of the BlackWatch solution, AppSensor was analyzed to gain an understanding of the strength and weaknesses of the tool. Four areas were considered:

- Implementation—*Is it feasible to implement AppSensor into an existing web application?*
- Configuration—*Can AppSensor be easily configured to suit the requirements of individual web applications?*
- Attack Detection—*Can AppSensor detect attacks while keeping false positives and negatives to a minimum?*
- Reporting—*Is the information gathered by AppSensor presented in an efficient manner?*

3.1.1. Implementation

As part of this research, AppSensor was implemented into MWR InfoSecurity’s internal application called “the Bazaare”. The Bazaare is an intentionally vulnerable ASP.NET web application used for training purposes within MWR. This type of application was required for this research as it contains many attack vectors and security flaws which AppSensor could be configured to monitor.

Communications between The Bazaare app and AppSensor used AppSensor’s Representational State Transfer (RESTful) API. RESTful communication uses HTTP requests to communicate data between applications. This approach requires minimal changes to be made to the Bazaare environment. However, integrating AppSensor with the Bazaare app presents a challenge. AppSensor does not provide client-side libraries allowing it to be integrated into web applications.

This meant the Bazaare web application itself had to be modified, ensuring it sent user events in a format which AppSensor could handle. To enable this, a custom class was developed within the Bazaare app which sent user events in the correct format using HTTP POST requests- events sent to AppSensor had to be in a specific format, including information such as: username, IP address, detection point (Figure 3 shows an example of data required by the AppSensor API [29]). Gaining an understanding of this structure and implementing this into an existing application highlights the challenges which developers face when attempting to increase attack awareness in their own web applications.

```

"user": {
  "id": "string",
  "ipAddress": {
    "address": "string",
    "geoLocation": {
      "id": "string",
      "latitude": 0,
      "longitude": 0
    },
    "id": "string"
  },
  "username": "string"
}
    
```

Figure 3. Example data required by the AppSensor API.

3.1.2. Configuration

The key attribute which makes AppSensor different to many other methods used to protect web applications, is its ability to work within the context of the web application itself. For AppSensor to achieve this contextual advantage it must be properly configured. This configuration stage was split into two main phases: identifying detection points and implementing response mechanisms. Identifying detection points is an essential step as it provides the areas where malicious activity can be identified. Implementing response mechanisms is what makes AppSensor effective at not only identifying malicious users, but also from stopping them exploiting the application.

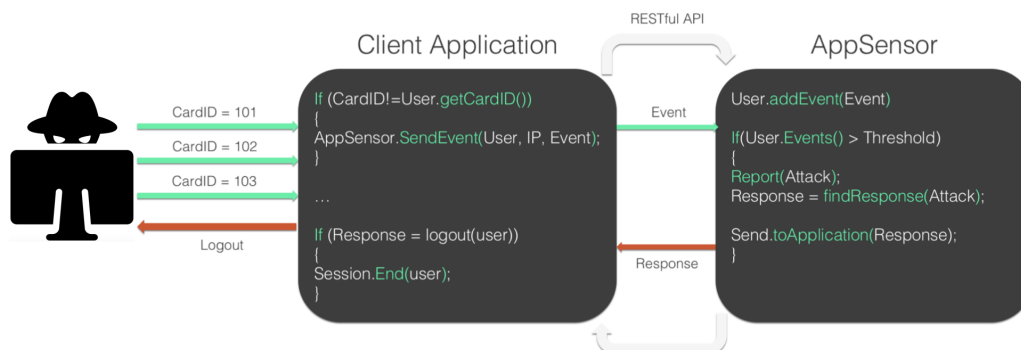


Figure 4. AppSensor CardID example.

Figure 4 (with malicious user image from [30]) demonstrates an example of how AppSensor works. Within this example the malicious attacker is altering the ‘CardID’ field of a POST request. If this vulnerability had not previously been remediated, then this would allow the attacker the ability to use a CardID that does not belong to them. However, because a detection point has been implemented at this stage, the Bazaar application checks to see if the CardID submitted matches the user’s card. If there is not match, the application discards the request and sends an event to AppSensor. AppSensor then uses information provided to determine if this activity exceeds the predetermined threshold that will identify this event as an attack. If an attack is identified, then AppSensor will alert the reporting mechanism and then provide the necessary response. In this scenario the response provided by AppSensor is to log the user out; however, AppSensor allows for configuration of incremental responses e.g., if the same user triggered this attack again within a certain time frame, the response could be increased in severity, for example, disabling the user’s account.

3.1.3. Attack Detection

One of the main benefits of AppSensor is that its attack detection capability is dependent on how the client application is configured. Providing developers with the ability to decide where malicious activity can occur within their application, allows for a high level of accuracy in detecting attacks. AppSensor uses a rule-based analysis mechanism to determine when user events have amounted to an attack. While this approach to attack detection can be very effective against basic attacks, it can be fairly easily bypassed by an experienced attacker that is aware of the security mechanisms in place.

3.1.4. Reporting

AppSensor provides a built-in web reporting portal where application activity can be monitored. This reporting mechanism displays information such as a live feed of events, attacks, and responses as well as displaying the current configuration of AppSensor. While this reporting mechanism is well presented, the underlying functionality is limited.

3.2. *Monitoring Unauthenticated Users*

Owing to the potential of attacks on web applications by malicious users, it is important to monitor unauthenticated users. Monitoring an authenticated user is simple— their actions can be associated with the account that they are performing certain activities under. Tracking the usage of users prior to authentication however is a far more challenging endeavor. The following section will discuss a variety of methods that have been investigated with regards to monitoring these unauthenticated users.

3.2.1. IP Addresses

One of the most common methods used for monitoring individual users throughout traditional monitoring tools—such as intrusion detection systems—is by monitoring the user's IP address. Monitoring IP addresses can be very effective in some circumstances, especially since IP addresses can be easily blocked at the networking layer. However, there are two main disadvantages of monitoring using this information: IP addresses are shared and they can be easily changed. As a result, of how IP address allocation works, an IP address does not necessarily only belong to one user, but may rather belong to a large group of users within the same network. As a result, blocking IP addresses may affect legitimate user functionality. Additionally, it is trivial for an experienced attacker to alter their IP address using a tool such as a virtual private network (VPN) [31].

3.2.2. Session Variables

Most web applications use session variables such as cookies or session identifiers to track users throughout the application. This method is one of the most widely adopted mechanisms for tracking application usage, as it can be very effective for monitoring individual users as each session variable is unique. However, the issue that accompanies web applications using this method, is that for this monitoring mechanism to work, users must willingly submit these session variables to the server. As a result, an attacker could simply strip these session variables from any communication they have with the server, and the web application would simply generate the user new session variables with each communication.

3.2.3. Device Fingerprinting

This information may include details such as: the client's operating system, the browser being used, plugins installed within the browser and the screen resolution of the device. This type of information is often gathered by web applications to provide a more tailored experience for the user, such as targeted marketing. However, this information may also be useful for identifying potentially malicious users. While many users will have similar fingerprints and hence be near impossible to distinguish using this approach, certain attributes may cause malicious users to 'standout'. For example,

most users are most likely going to be using common operating systems such as: Windows, Mac OS, Android, and iOS. This means that if a user device is running an operating system such as Kali Linux (an operating system designed for penetration testers) then this user may be identified as ‘suspicious’ and therefore their actions should be monitored closely [32].

Factors outlined in this section must be considered when designing a solution which aims to increase attack awareness.

3.3. Design

A solution was developed that allows applications the ability to identify attacks from within the context of the application; and to respond effectively to deter attackers. The development process of this solution was influenced by two important factors: application scalability and well documented and structured code.

One of the first stages of the development process was to design the structure of the application. The structure diagram shown in Figure 5 demonstrates the initial design of this proposed solution—hereby named ‘BlackWatch’.

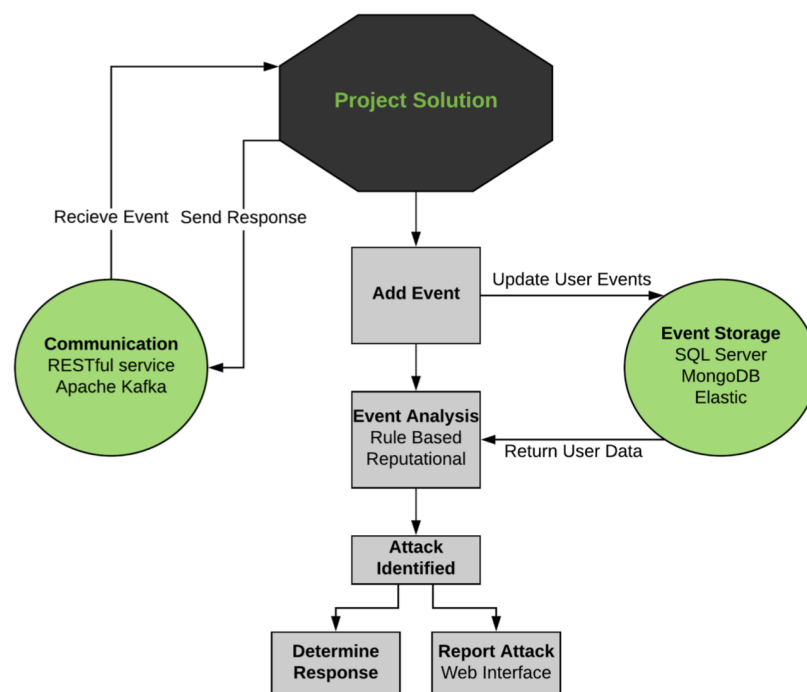


Figure 5. BlackWatch system overview.

3.4. Implementation

BlackWatch was implemented in the Python programming language. One of the first stages of the development process was to design the structure of the application. The first step taken within the implementation stage was to develop a RESTful API service that would allow for communication between the client application and the BlackWatch solution. A RESTful server uses HTTP requests for communicating with clients, the HTTP protocol uses several different methods for sending and receiving data; for the BlackWatch solution the methods used are POST and GET requests. Using Flask, this type of functionality can be easily implemented as shown in Figure 6, where a POST request will be received and then processed in attempt to retrieve information regarding an application event.


```
@app.route('/addevent', methods = ['POST'])
def addEvent():
    try:
        event = request.data.decode('utf-8')
        Result = ParseEvent(event)
    except:
        Result = "Invalid formatting"
    return Result
```

Figure 6. RESTful server—POST method.

The remaining components that had to be implemented into the BlackWatch application were split into four general areas; handling events, identifying attacks, determining responses, and reporting.

3.4.1. Handling Events

During the first stage, handling events, it is essential to ensure that the event received from the client application contains all of the necessary information in the correct format. Each event must contain two main objects: 'User' and 'Detection Point', the first object contains information identifying the user responsible for the event, and the second where the event was triggered. This information must be present—and correctly formatted—in order for the event to proceed further within the application. Given that this information is correct, the event must then be added to the BlackWatch database and sent for analysis.

BlackWatch uses the MongoDB database solution [33]. PyMongo allows the BlackWatch application the ability to add and retrieve information from the database with ease, as well as including functionality for performing filtered queries to retrieve relevant information from the database quickly. The documents stored within MongoDB are done so in a Binary JSON format (BSON); very similar to the JSON format used throughout the BlackWatch application.

3.4.2. Identifying Attacks

Once an event has been added to the BlackWatch database, the next step is to send this event through the analysis process. The purpose of the analysis process is to identify whether an event holds enough merit to be identified as an attack. There are two main analysis mechanisms within the BlackWatch solution used to determine this; rule-based and reputation-based analysis. Rule-based analysis is a very simple method of attack identification that has been used within security mechanisms for years. The idea being that if a user triggers a predetermined number of events within a specific area during a certain period of time, then these events are identified as an attack and a response is set accordingly. For example, if the user 'Bob' triggers an event at the 'Login Page', then this does not necessarily mean that Bob is performing malicious activity, it may simply be a mistake. However, if Bob were to trigger this event ten times within 30 seconds, then it is highly likely that this is an attack and hence the application should respond. This type of attack detection can be very effective; however, its main downfall is that it does not take into consideration a user's activity throughout other areas of the application.

When rule-based analysis does not identify an event as an attack, the reputation-based analysis mechanism is invoked. The reputation-based analysis mechanism analyzes a users' behavior over the entire application within a certain time frame. The idea being that while a user has not triggered enough events in one specific area to trigger an attack, they may have shown a pattern of malicious activity across the application. For this mechanism to work, each detection point is given a level of severity. For example, a user inserting special characters into a search function could very easily be

a mistake, hence this detection point will be given a ‘Very Low’ severity rating. However, a user uploading a malicious file to an application is almost certainly an attack and hence should be given a ‘High’ severity rating. These severity ratings hold a certain weighting within the BlackWatch application as shown in Table 1.

Table 1. Severity ratings and values.

Severity Rating	Value
High	8
Medium	4
Low	2
Very Low	1

The values of the severity weighting increase by powers of two, representing an incremental structure whereby high must have a greater value than low. Every time a user triggers an event, the severity weighting belonging to the event’s detection point will be added to the user’s reputation. Once a users’ reputation exceeds nine, the user is identified as malicious and an attack is triggered. One of the most important features of the reputation-based analysis mechanism is that a users’ reputation must decrease over time; this ensures that if an attacker stops abusing the application, their reputation must reflect their legitimate behavior. To achieve this feature, the application will decrease the user’s reputation by one every ten minutes until the reputation is equal to zero.

One of the main advantages of each of the mechanisms discussed above, is that both can monitor unauthenticated users. Both mechanisms will check to see if a username has been provided for the user in question, and if not will handle the analysis based on the user’s session ID. This approach allows BlackWatch the ability to monitor unauthenticated users throughout the application, and if necessary to respond to their malicious activity.

3.4.3. Determining Responses

Upon successful identification of an attack, the necessary response must be determined. The BlackWatch application allows users the ability to set multiple responses for the same detection point. This allows for the responses chosen to be done so in an incremental manner; meaning that if a user has triggered an attack at this same detection point within the past 30 minutes, then the response chosen can be incremented to take more severe action. Once the necessary response has been identified, it is then added to the ‘Responses’ MongoDB collection. Client applications can access these responses using an HTTP GET request containing the intended username and session ID.

The BlackWatch application will return any responses specific to that user and then delete these responses from the database. This is to prevent any duplicate responses from being returned to the client application. The reason that responses are sent to the client application in this manner and not simply returned in response to the initial event sent from the client, is that this approach ensures that the client application’s performance is not affected while the BlackWatch application analyzes the event. Once responses are sent to the client application, it is then up to that application to execute these responses using its own functionality. The main benefit that arises from responses being executed in this manner, is that the only limitations to what responses can be set, are bound by the functionality of the application itself.

At this stage of the development process the BlackWatch application contains the ability to: receive suspicious events from client applications, analyze these events to determine if they indicate an attack, and if so identify the necessary response. While this functionality is what makes the BlackWatch solution effective at deterring malicious attackers, the overall objective of this application is to raise attack awareness within web applications and hence this functionality must provide a well-structured reporting mechanism.

3.4.4. Reporting

Rather than simply reporting BlackWatch's activity using log files like so many other security solutions, this application uses a web interface to display the relevant information in a professional and efficient manner. One of the main challenges that faced this stage was the decision on how to communicate between the web application client (the reporting mechanism) and the web server (the BlackWatch application). For the reporting mechanism to reflect the live activity that is happening within the BlackWatch application, it was essential for the web server to be able to send events directly to the reporting application. However, there lies a difficulty in that traditionally web applications only communicate with the server when they need to, and hence the server has no way of sending information unless it is first requested [34].

To tackle this issue WebSocket technology was implemented. WebSockets are a method used for creating bidirectional communication between client-side applications and web servers. JavaScript has a standard library called Socket.IO which is used within the web reporting application. The Flask library Flask-SocketIO is used within the BlackWatch application to allow the server to implement WebSocket technology, and to send information to the web application on demand [35].

As well as displaying the activity that is taking place within the BlackWatch application (as demonstrated in Figure 7), the reporting mechanism can also be used for configuring detection points to be used within your application. The ability to create and remove detection points from within the web reporting mechanism makes it easy for users to configure the BlackWatch solution to fit the needs of their web application.

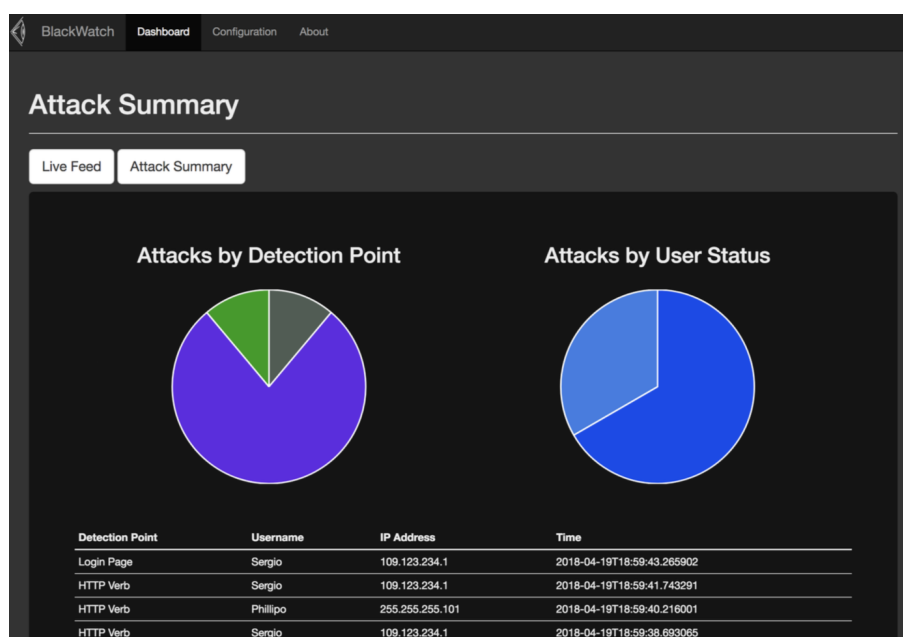


Figure 7. The Attack Summary interface within BlackWatch.

Following the development of the BlackWatch solution, an evaluation was performed. The success of BlackWatch is based on the coverage of four main areas: ease of implementation, ability to identify unauthenticated users, accuracy of attack detection, effectiveness, and efficiency of reporting. The subsequent section highlights the results, before providing an analysis in Section 5.

4. Results

As part of the evaluation, participants were asked to use BlackWatch with the Damn Vulnerable Web Application (DVWA) developed by Dewhurst Security [36]. This is a web application containing several security flaws, and presents a suitable testbed for evaluating BlackWatch.

4.1. Unit Tests

The unit tests created during the design stage of this research were developed to evaluate the BlackWatch application's ability to: receive events, detect attacks using both rule-based and reputation-based analysis and to retrieve responses from the application.

The BlackWatch application passed all unit tests (highlighted in green in) Table 2, except for the response retrieval test; this test was designed to trigger an attack and to then try and retrieve the correct response from the BlackWatch application. During the development process the approach used to communicate responses changed, therefore causing this unit test to fail, (highlighted in red in) Table 2. Despite failing this unit test, the overall results from these tests were positive, demonstrating that BlackWatch could not only detect attacks by authenticated users, but also by monitoring unauthenticated users with session identifiers as discussed in Section 3.2. Table 2 demonstrates the results of the unit testing phase.

Though BlackWatch has not yet been released publicly, these results are reproducible while using the freely available Damn Vulnerable Web Application [37]. Furthermore, similar results are to be expected when BlackWatch is run against existing web applications.

Table 2. BlackWatch unit test results.

Unit Test	Result	Intended Result	Pass or Fail
Send Correctly Formatted Event	"Event is being added"	"Event is being added"	PASS
Send Invalid IP Address within Event	"Incorrect formatting within POST request"	"Incorrect formatting within POST request"	PASS
Detect Attack using Username	Attack shown within the reporting mechanism	Attack shown within the reporting mechanism	PASS
Detect Attack using Session ID	Attack shown within the reporting mechanism	Attack shown within the reporting mechanism	PASS
Detect Attack using Reputation-Based Analysis	Attack shown within the reporting mechanism	Attack shown within the reporting mechanism	PASS
Retrieve Response	"Failed"	"Response Returned"	FAIL

4.2. User-Based Evaluation

A total of five participants took part in stage of the evaluation. The number of participants may seem relatively small however the evaluation process took over an hour to complete, and individuals participated on a voluntary basis. Therefore, due to time constraints it was only feasible to include a small sample size during this stage.

The results gathered from the user-based evaluation process are split into four main sections: participant skill-set, BlackWatch setup and configuration, attack detection, and reporting. These results consist of both quantitative and qualitative data.

4.2.1. Participant Skillset

The evaluation process considered the development skillset possessed by participants. Participants were asked to rate their overall level of development experience on a scale of 1 to 5 (whereby 5 equaled Very Experienced) as part of a questionnaire (Figure 8). Overall, results showed participants considered themselves to have an average level of development experience ($n = 5$, $mean = 3.2$).

Participants were also asked to rate their development knowledge with both PHP, and Python. Overall results showed that while participants considered themselves to have an average level of development knowledge in Python ($n = 5$, $mean = 3.2$, $mode = 3$), they generally felt they had a lower level of knowledge in PHP ($n = 5$, $mean = 1.6$, $mode = 1$).

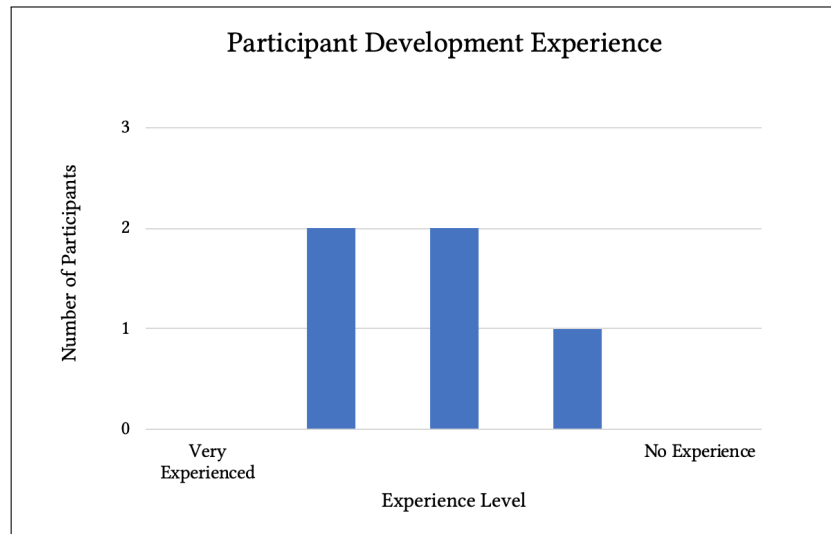


Figure 8. Development experience of participants.

4.2.2. BlackWatch Setup and Implementation

The questions within this section aimed to identify participants' experience during the setup and implementation of the BlackWatch solution. The practical steps required during this stage were guided by a tutorial provided by the BlackWatch solution; all participants stated that these instructions covered all the necessary areas throughout the setup procedure. 80% of participants rated both the BlackWatch setup and the client-side implementation as being 'very easy' and 20% rated it as 'easy' ($n = 5$, $mean = 4.8$, where five equaled very easy). None of the participants recorded any difficulties faced throughout this stage and no recommendations were made in regards to improving the instructions provided.

4.2.3. Attack Detection and Reporting

Upon taking on the role of a malicious attacker targeting the web application, participants rated the BlackWatch applications accuracy of attack detection as having a mean score of 4.8—one being not accurate, five being very accurate—meaning that the malicious activity they were performing was being detected. A mean score of 4.8 was also given to the effectiveness of the responses provided by the BlackWatch application—one being not effective, five being very effective. Within this section participants were asked whether or not they believe that the BlackWatch application would "deter attackers from targeting that application", every participant responded 'Yes'. While the most important aspect of this research was the BlackWatch application's ability to detect attacks, it was also essential that the application provided a well-structured reporting mechanism. Participants were asked to rate their experience using the web reporting mechanism, 80% of these participants agreed that the mechanism provided was very efficient with one participant rating the mechanism's efficiency as 4 out of 5—one being not efficient, five being very efficient ($n = 5$, $mean = 4.8$).

4.2.4. Overall

One of the key questions asked throughout this evaluation process was whether participants believed that if properly implemented, the BlackWatch solution would increase attack awareness within web applications. Every participant that took place within this evaluation agreed 'Yes'. The final section of the survey asked participants for any additional comments that they have relating to the BlackWatch application; most comments given during this stage praised the design and usability of the reporting mechanism used by the BlackWatch application.

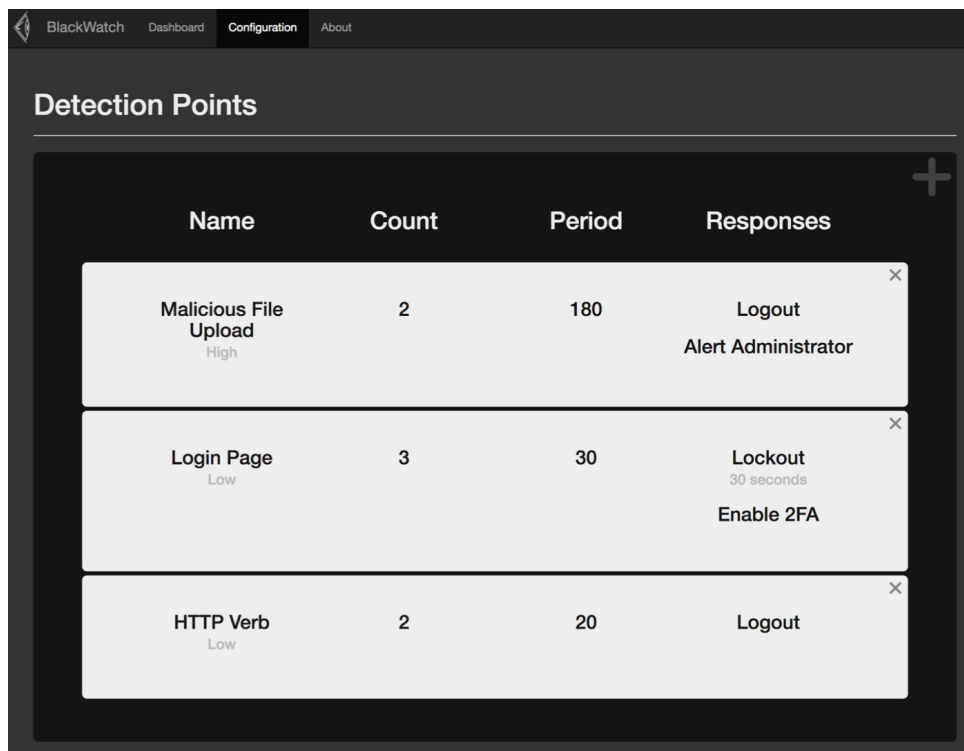
5. Discussion

5.1. Ease of Use

It is vital for the BlackWatch solution to be considered user-friendly. For this type of solution to attract developers and security professionals, it must be easy for them to understand and implement. Setting up this solution proved to be simple, even for participants that rated their development experience at the lower end of the experience scale still managed to setup and implement the BlackWatch solution with ease.

Interestingly, the results demonstrated in Section 4.2.1 indicate that the participants who took part in the evaluation process felt far more knowledgeable of the Python programming language than they did with PHP. This was one of the main reasons that this programming language was chosen, as Python's easy to learn syntax ensures that developers interested in using this solution, can review the underlying code in attempt to understand the full functionality of the BlackWatch solution.

The area that requires the most effort from those implementing this solution, is the configuration of detection points within the client application. Often security solutions are based outwith the applications they protect. However, the attribute that makes the BlackWatch solution effective at identifying attacks, is that it works within the context of the application. Developers have a clear understanding of the legitimate usage of their applications and hence they should be able to accurately determine what classifies as malicious, or suspicious behavior. Therefore, to implement the BlackWatch solution into their application (Figure 9), developers simply need to identify points where they can detect this malicious behavior and insert the necessary code to communicate with the BlackWatch solution.



Name	Count	Period	Responses
Malicious File Upload <small>High</small>	2	180	Logout Alert Administrator
Login Page <small>Low</small>	3	30	Lockout <small>30 seconds</small> Enable 2FA
HTTP Verb <small>Low</small>	2	20	Logout

Figure 9. Configured detection points interface in BlackWatch.

The evaluation of the proposed application made use of a pre-written PHP library that allowed users to communicate very easily with the BlackWatch solution, this client-side library could be easily replicated into other programming languages as discussed within the future work section. During the evaluation procedure participants found the creation of detection points to be easy, even though they were not familiar with the client application they were working with. This indicates that the

implementation process should be very easy for a developer, as they will have prior knowledge of the web application environment they are working within.

One concern that surrounds this approach of attack detection within web applications, is that for a developer to know where to implement detection points within their application, they must be aware of the type of attacks a malicious user may perform.

While this may be an easy task for developers with security experience, it may pose as a difficulty for those who have never performed any research into the offensive side of security. A possible action that could be taken to counteract this issue, would be to involve a security professional within the implementation stage of the BlackWatch solution. A security professional, such as a penetration tester, would be able to advise on the areas that an attacker is likely to attack, and how these attacks could be identified with a high degree of accuracy.

5.2. Attack Detection

OWASP [28] states that “*exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident*”. As discussed previously, a lack of attack detection can leave an organization vulnerable to advanced persistent threats. During the evaluation of BlackWatch, participants were asked to portray the role of a malicious attacker attempting to exploit the DVWA.

BlackWatch monitored participants’ activity to identify attacks and respond accordingly. The results from this section indicate that the malicious activity performed by each participant was being detected to a high degree of accuracy.

5.2.1. False Positives and Negatives

When developing a security product one of the main concerns that must be addressed is the occurrence of false positives and negatives. False positives occur when legitimate behavior is wrongly identified as an attack. False negatives are the exact opposite—when an attack is wrongly identified as legitimate behavior and hence no action is taken to reprimand the attacker. False positives are often the more severe issue for organizations as they can affect legitimate users and negatively impact user experience.

During the evaluation of this solution, false positives were not found to be an issue as the detection points created within the client web application took user error into consideration. For example, within the client application used there was a component that could be used to ‘ping’ an IP address. Functionality such as this would often be tested by malicious users for vulnerabilities such as command injection [38]. Hence, a detection point was configured here to detect any input that does not match the application’s expectations. As an attempt to reduce false positives, if the user input matches the correct format of an IP address, but is not a legitimate address e.g., ‘455.455.1.1’ (each octet must be below 256) then no event is sent to the BlackWatch application, as this could have been a user error.

The reputation-based analysis mechanism within the BlackWatch application is used as an attempt to reduce false negatives, as it attempts to correlate events that are being triggered throughout the application by individual users. The unit testing performed during the evaluation of the BlackWatch solution demonstrated the application’s ability to identify attacks based on a pattern of events occurring throughout different components of the client application, rather than the abuse of one single detection point. This detection technique can help identify attackers who may be attempting to avoid the rule-based detection mechanism. However, this approach also introduces the risk of producing false positives. With rule-based detection it is easy to identify an attack based on the repetition of one suspicious event. However, with reputation-based analysis it is more likely that a legitimate user could trigger several different ‘suspicious’ events throughout multiple areas of the application through user error. Despite this increased risk, none of the participants within the evaluation process experienced any false positive or false negative errors.

5.2.2. Monitoring Unauthenticated Users

It was essential for this application that there was some mechanism in place for monitoring unauthenticated users and their actions. Discussed in Section 3.2 are numerous different approaches that can be used to monitor users who have yet to authenticate within the client application.

One of the most common approaches security solutions take to monitor these users is by tracking their IP address; however, due to this approach’s lack of accuracy—and potential implications to legitimate users—another approach was taken instead.

Session variables such as session IDs are provided to all users of an application as a means of ensuring that the content they see is specific to their individual session. The results from the evaluation stage demonstrate that this session variable can be used as a method of user monitoring, and can effectively associate attacks with individual users. However, despite the effectiveness of this method against novice attackers, an experienced attacker would most likely be able to figure out that their session ID is being used to monitor their attacks, and hence they could quite easily remove this information from the requests they send to the web application. Even though it is possible for an attacker to bypass this method of activity monitoring, it is still an effective approach that adds an extra layer of security to the web application.

5.2.3. Reporting

The reporting mechanism provided with the BlackWatch application allows administrators the ability to monitor the events and attacks occurring within their application (Figure 10). According to the results from the evaluation stage, participants found the reporting mechanism to be very well designed and user-friendly. Throughout the research the reporting mechanism developed was created with the intention of being simple and efficient; however, given further development there are many features that could be added to improve this aspect of the BlackWatch application.

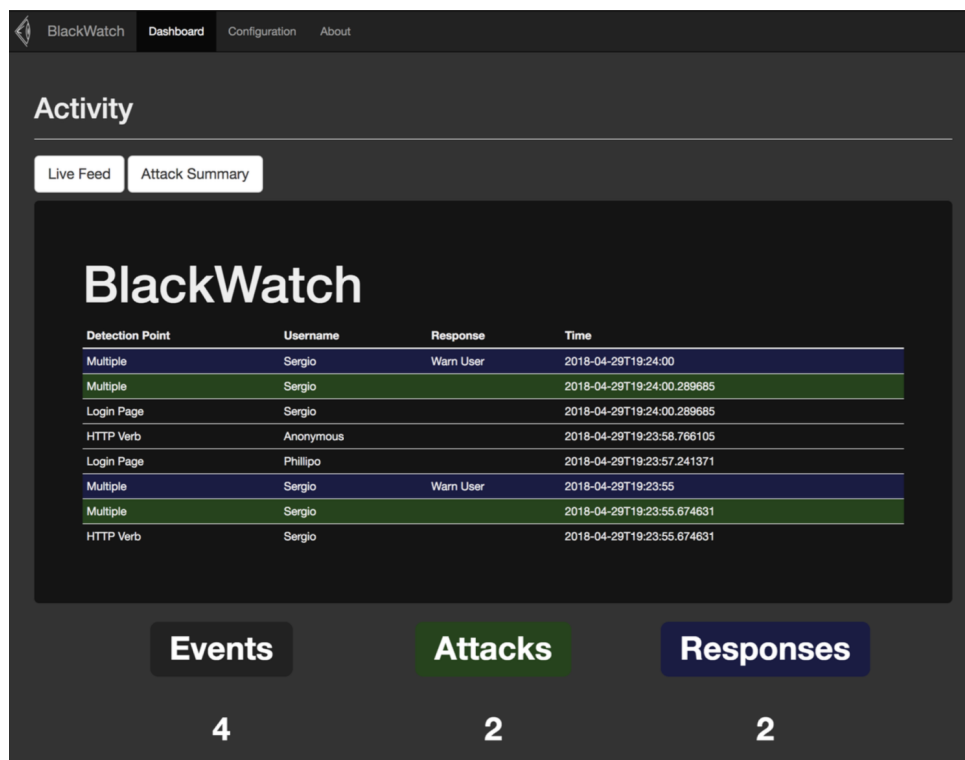


Figure 10. BlackWatch dashboard interface.

5.3. Attack Response

Having the ability to accurately detect malicious users within an application gives developers the ability to respond with deterring action. Giving applications the ability to detect attacks was the main objective of this research. It was also important to investigate methods that could be used to respond to these attacks. During the evaluation of the application the main responses used were:

- User Warning
- User Logout
- Page Redirect
- Fake Output (such as mock database) information

The ‘fake output’ was a response used at a detection point aimed to identify SQL injection attacks against the client web application being tested. The purpose of this response was to provide false information to the attacker to confuse, and or, waste their time. Participants involved during the evaluation stage all stated that they believe these responses would successfully deter or slow down an attacker. As stated within the methodology stage of this project, one of the key aspects of how the BlackWatch application operates, is that the responses that can be used are only limited by the client web application’s functionality.

5.4. Challenges

During the initial research it was apparent that this approach to attack detection was still very much a concept that had yet to be exposed to the security landscape. This introduced several difficulties as it was often challenging to find the relevant literature to help guide the development.

The most challenging task involved with this approach of attack detection is being able to encourage developers to take the extra step of implementing this functionality into their application. While the results from the evaluation stage indicate that the BlackWatch solution is easy to setup and implement, it is still an extra step that a developer would need to take within the already complex process of developing an application. However, as discussed in Section 2.3 implementing security characteristics such as attack awareness into an application during the development process can help prevent security incidents and hence save organizations both time and resources in the long run.

6. Conclusions and Future Work

This research demonstrates that it is no longer enough for organizations to focus their efforts towards preventing attacks, but rather stresses the importance of detecting the presence of these attacks in the first place. Being able to accurately detect attacks within a web application can allow organizations the ability to execute responses with the objective of detecting, or slowing down attackers.

The methods currently used for detecting attacks within the cyber community often lack the necessary configuration required to produce effective detection rates and hence attacks often go unnoticed.

The BlackWatch application presented here attempts to increase the levels of attack awareness by working within the context of web applications themselves. By implementing points within applications where malicious activity can be detected, developers are given the ability to confidently identify user behavior that does not appear as legitimate usage. This behavior is then correlated outwith the web application itself to determine if a user has demonstrated a pattern of malicious behavior. If such an attack is identified, then the appropriate response is determined.

A concern that accompanied this research and its approach to attack detection, was the challenge of encouraging developers to implement this type of security into their application. To ensure this issue did not hinder the research objective of increasing attack awareness, the BlackWatch application was developed with ease of use being one of the key aspects. Through a preliminary user-based evaluation it was demonstrated that not only is the BlackWatch application easy to setup, but also it is also easy

to configure within a client environment. One of the factors that contributes to this ease of use is that developers are implementing the solution into their application—a programming environment they fully understand. It is this usability factor combined with the BlackWatch solutions' accuracy of attack detection that highlights the success of this research in providing a method of increasing attack awareness within web applications.

When developing the BlackWatch solution, the objective was to create a proof of concept. However, it is essential that upon further testing and research into this area that new analysis mechanisms are introduced to continually improve the accuracy of attack detection. The web reporting mechanism used by the BlackWatch application could also be improved by implementing more advanced components e.g., during the design stage of this application the idea of manual responses was considered; manual responses would allow an administrator to manually set the responses for malicious users from within the reporting mechanism.

To ensure that the BlackWatch application can be widely used throughout a variety of web applications, it is essential that more client libraries are developed to allow the BlackWatch solution to be easily implemented into the most common web application frameworks.

An area that was not covered during this research, was the deployment of the BlackWatch solution within a production environment. The objective of this work was to demonstrate the level of effectiveness provided by this method of attack detection. However, the solution was only evaluated within a test environment, and hence its performance has never been tested within a production scenario. Prior to deploying the BlackWatch application alongside a real-world application, developers must firstly investigate several different areas to ensure that the BlackWatch application performs in an efficient, effective, and reliable manner. One of the main areas that must be researched is the server used to host the BlackWatch application. The server used to host the BlackWatch application will be one of the main factors that determine how well the application performs, and hence it is crucial that future work involving server selection is carried out to help guide developers.

Author Contributions: Investigation, C.C.H.; Supervision, L.A.S. and N.C.; Writing—original draft, C.C.H.; Writing—review & editing, L.A.S. and N.C.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Riek, M.; Bohme, R.; Moore, T. Measuring the influence of perceived cybercrime risk on online service avoidance. *IEEE Trans. Dependable Secure Comput.* **2016**, *13*, 261–273. [CrossRef]
2. Kumar, S.; Mahajan, R.; Kumar, N.; Khatri, S.K. A study on web application security and detecting security vulnerabilities. In Proceedings of the IEEE 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 20–22 September 2017; pp. 451–455.
3. Rodriguez, C.; Martinez, R. The Growing Hacking Threat to Websites: An Ongoing Commitment to Web Application Security. In *A Frost & Sullivan White Paper*; Frost and Sullivan: San Antonio, TX, USA, 2012.
4. Verizon. Data Breach Investigations Report. 2018. Available online: https://www.verizonenterprise.com/resources/reports/rp_DBIR_2018_Report_en_xg.pdf (accessed on 17 November 2018).
5. Shepherd, L.A.; Archibald, J.; Ferguson, R.I. Perception of risky security behaviour by users: Survey of current approaches. In *Human Aspects of Information Security, Privacy, and Trust: First International Conference, HAS 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21–26, 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 176–185.
6. Mathew, S.; Britt, D.; Giomundo, R.; Upadhyaya, S.; Sudit, M.; Stotz, A. Real-time multistage attack awareness through enhanced intrusion alert clustering. In Proceedings of the Military Communications Conference (MILCOM 2005), Atlantic City, NJ, USA, 17–20 October 2005; Volume 3, p. 1801.
7. Ben-Meir, E. Prevention vs. Detection in Cyber Security: Why Not Both? 2017. Available online: <https://blog.cyberint.com/prevention-vs-detection-in-cybersecurity-why-not-both> (accessed on 17 November 2018).

8. MWRInfoSecurity. Tips for Success When Building a Detection Capability. 2018. Available online: <https://www.mwrinfosecurity.com/our-thinking/building-attack-detection-capability/> (accessed on 17 November 2018).
9. PCISSC. Payment Card Industry Data Security Standard—Requirements and Security Assessment Procedures. 2016. Available online: https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2.pdf (accessed on 17 November 2018).
10. High-Tech Bridge Security Research. Patching Complex Web Vulnerabilities Using ModSecurity WAF. 2016. Available online: <https://www.htbridge.com/blog/patching-complex-web-vulnerabilities-using-modsecurity-waf.html> (accessed on 19 October 2018).
11. Kolochenko, I. Web Application Firewall: A Must-Have Security Control or an Outdated Technology? 2016. Available online: <https://www.csoonline.com/article/3032743/application-development/web-application-firewall-a-must-have-security-control-or-an-outdated-technology.html> (accessed on 19 October 2018).
12. Ahmed, M. Evading All Web-Application Firewalls XSS Filters. 2015. Available online: <https://mazinahmed.net/uploads/Evading%20All%20Web-Application%20Firewalls%20XSS%20Filters.pdf> (accessed on 17 November 2018).
13. Clincy, V.; Shahriar, H. Web Application Firewall: Network Security Models and Configuration. In Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 23–27 July 2018; Volume 1, pp. 835–836. doi:10.1109/COMPSAC.2018.00144. [CrossRef]
14. Jones, R.L.; Rastogi, A. Secure coding: Building security into the software development life cycle. *Inf. Syst. Secur.* **2004**, *13*, 29–39. [CrossRef]
15. Goertzel, K.M.; Winograd, T. *Enhancing the Development Life Cycle to Produce Secure Software*; US Department of Defense: Arlington, VA, USA, 2008; pp. i–iv.
16. Hoff, J. 6 Ways to Strengthen Web App Security. 2012. Available online: <https://www.darkreading.com/risk-management/6-ways-to-strengthen-web-app-security/d/d-id/1106197> (accessed on 17 November 2018).
17. Haridas, N. Software Engineering—Security as a Process in the SDLC. 2007. Available online: <https://www.sans.org/reading-room/whitepapers/securecode/paper/1846> (accessed on 17 November 2018).
18. Hasan, A.M.; Meva, D.T.; Roy, A.K.; Doshi, J. Perusal of web application security approach. In Proceedings of the 2017 International Conference on Intelligent Communication and Computational Techniques (ICCT), Jaipur, India, 22–23 December 2017; pp. 90–95.
19. Futch, L.; von Solms, R. SecSDM: A model for integrating security into the software development life cycle. In *Fifth World Conference on Information Security Education*; Springer: Boston, MA, USA, 2007; pp. 41–48.
20. Karim, N.S.A.; Albuolayan, A.; Saba, T.; Rehman, A. The practice of secure software development in SDLC: An investigation through existing model and a case study. *Secur. Commun. Netw.* **2016**, *9*, 5333–5345. [CrossRef]
21. Gregory, P.H. Security in the Software Development Life-Cycle. 2003. Available online: <https://searchsecurity.techtarget.com/tip/Security-in-the-software-development-life-cycle> (accessed on 7 January 2019).
22. Razzaq, A.; Hur, A.; Shahbaz, S.; Masood, M.; Ahmad, H.F. Critical analysis on web application firewall solutions. In Proceedings of the 2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS), Mexico City, Mexico, 6–8 March 2013; pp. 1–6.
23. Singh, J.J.; Samuel, H.; Zavarisky, P. Impact of Paranoia Levels on the Effectiveness of the ModSecurity Web Application Firewall. In Proceedings of the IEEE 2018 1st International Conference on Data Intelligence and Security (ICDIS), South Padre Island, TX, USA, 8–10 April 2018; pp. 141–144.
24. Krueger, T.; Gehl, C.; Rieck, K.; Laskov, P. TokDoc: A self-healing web application firewall. In Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 22–26 March 2010; pp. 1846–1853.
25. OWASP. OWASP AppSensor. 2017. Available online: https://www.owasp.org/index.php/OWASP_AppSensor_Project (accessed on 17 November 2018).
26. Thomassen, P. AppSensor: Attack-Aware Applications Compared Against a Web Application Firewall and an Intrusion Detection System. Master’s Thesis, Institutt for Datateknikk og Informasjonsvitenskap, Trondheim, Norway, 2012.
27. OWASP. OWASP Top Ten Project. 2010. Available online: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (accessed on 17 November 2018).

28. OWASP. Top 10 2017. 2017. Available online: https://www.owasp.org/index.php/Top_10-2017_Top_10 (accessed on 17 November 2018).
29. OWASP. Appsensor-Ws-Rest-Server. Available online: http://appsensor.org/docs/v2.3.0/api/ui/index.html#!/RestControllerHandler/resource_RestRestControllerHandler_addEvent_POST (accessed on 1 February 2019).
30. icons8. Hacker Icon. Available online: <https://visualpharm.com/free-icons/hacker-595b40b75ba036ed117d616b> (accessed on 1 February 2019).
31. Belesi, R. How to Change Your IP Address. 2016. Available online: <https://blogs.opera.com/news/2016/09/how-to-change-ip-address/> (accessed on 17 November 2018).
32. Abouollo, A.; Almuhammadi, S. Detecting Malicious User Accounts Using Canvas Fingerprint. In Proceedings of the 2017 8th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, 4–6 April 2017; pp. 358–361.
33. MongoDB. 2018. Available online: <https://www.mongodb.com/> (accessed on 19 October 2018).
34. West, M. An Introduction to Websockets. 2018. Available online: <http://blog.teamtreehouse.com/an-introduction-to-websockets> (accessed on 7 April 2017).
35. Grinberg, M. Flask-Socketio. 2018. Available online: <https://github.com/miguelgrinberg/Flask-SocketIO> (accessed on 19 October 2018).
36. Dewhurst, R. Dewhurst Security—Professional Security Services. 2017. Available online: <https://dewhurstsecurity.com/> (accessed on 1 February 2019).
37. DVWA. Damn Vulnerable Web Application (DVWA). 2018. Available online: <http://www.dvwa.co.uk/> (accessed on 19 October 2018).
38. OWASP. Command Injection. 2016. Available online: https://www.owasp.org/index.php/Command_Injection (accessed on 19 October 2018).



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).