Cheat Detection In Cyber Security Capture The Flag Games - An Automated Cyber Threat Hunting Approach

Robert A. Chetwyn[1], László Erdődi[2]

[1]University of Oslo, Department of Informatics, Gaustadalléen 23B, 0373 Oslo, Norway
[2]Norwegian University of Science and Technology, Department of Information Security and Communication Technology, Gløshaugen, 7034 Trondheim, Norway

Abstract

Capture-the-flag style cyber security games (CTF) are one of the most popular ways of learning and teaching ethical hacking. These CTF games usually present a set of hacking tasks or challenges that simulate a vulnerability to be compromised. When the participant compromises the vulnerability they are presented with a secret flag that is uploaded to prove a participants completion of a challenge. Whilst this secret flag confirms successful completion of a challenge, it does little to verify the legitimacy of a participant's activities. We propose a process for plagiarism detection in web application CTF games via automated cyber threat hunting techniques. Using log data captured from penetration testing courses, we develop a series of indicators of compromise for each CTF challenge that are attributed to a participant's activities. We develop an automated querying tool that interfaces with the Elastic Stack to query these IOCs for classifying participant activities as suspicious or benign without false positives.

Keywords

Security automation, threat hunting, security education, plagiarism, penetration testing

1. Introduction

Capture-the-flag style cyber security games (CTF) are one of the most popular ways of learning and teaching ethical hacking. These CTF games usually present a set of hacking tasks or challenges that simulate a vulnerability to be compromised such as in [1] [2] [3] and our own CTF platform: Hacking Arena [4]. When the participant compromises the vulnerability they are usually presented with a secret flag that is used to prove a participant's successful completion of a challenge. Whilst this secret flag confirms successful completion of a challenge, it does little to verify the legitimacy of this compromise.

This lack of verification is a problematic scenario in both academic and industry environments, where plagiarism affects the integrity of the provided courses and participants certification. In 2019 plagiarism was reported for the Offensive Security Certified Professional (OSCP) exam where an ex-participant produced public write-ups on the OSCP exam challenges, leaking the exam solutions [5]. These compromised challenges were still reported present in later

C&ESAR'21: Computer Electronics Security Application Rendezvous, November 16-17, 2021, Rennes, France
roberac@ifi.uio.no (R. A. Chetwyn); laszlo.erdodi@ntnu.no (L. Erdődi)
0000-0002-2028-849X (R. A. Chetwyn); 0000-0002-4910-4228 (L. Erdődi)

examinations after the leak, consequently leading to possible reuse of challenge solutions. Similarly, examination 'brain-dumps' - the publishing of exam questions, topics and answers [6] create a problem with information reuse. Participants can reuse the information provided from brain-dumps to complete CTF challenges, skipping pre-requisite steps and submitting the flags.

With the popularity of using CTF challenges for delivering cyber security training we are motivated to ensure the integrity of this delivery and to monitor each challenge for plagiarism. This paper provides an automated monitoring solution for web based CTF challenges based upon cyber threat hunting that is capable of detecting plagiarism activities, with high precision.

The paper is structured as follows: Section 2 provides background information into CTF challenges, an overview of the Hacking Arena and the proposed threat hunting methods. Section 3 discusses related research. Section 4 explains the infrastructure and methods used to conduct the research. Section 5 presents the findings and Section 6 discusses these findings in depth.

2. Background

CTF competitions usually present a set of hacking tasks or challenges where each challenge is defined by one vulnerability or a chain of vulnerabilities associated with a secret flag. The aim of a participant is to exploit the vulnerability in each challenge, and thus capture the associated flag. Once captured, the participant submits this flag as confirmation of CTF completion. Unlike in real hacking scenarios no further steps are required from the attacker after exploiting the vulnerability such as maintaining continuous access to the target, uploading attacking scripts, or establishing a connection to a command and control server.

Based on the secret flag an unambiguous criterion is provided for each challenge to decide whether a challenge was solved or not. Challenges can be classified according to the type of problem they present (e.g., web hacking challenge or binary exploitation). Unlike in real security incidents the human factors are excluded from the solution unless other information is provided for the challenge, so an attacker has to rely on their knowledge and reasoning, but not on social engineering. To find the solution of the hacking challenge the participants have to carry out attack steps in the right order. By solving a step the participants might receive new information to achieve the next step. Additional information can come from challenge hints (e.g. what is worth trying) too provided for the challenge to help the attacker to proceed in the right direction.

Standard CTFs run in Jeopardy mode, meaning that all the participants are attackers, and all the challenges are static, so challenges are not changing throughout the competition. In other CTF variants, participants may be subdivided to two teams:a red team, focusing on attacking the target system, and a blue team for defending the target system. Alternatively, each team may be provided with an infrastructure they have to protect - they can change it to strengthen the service - while, at the same time, attacking the infrastructure of other teams. Considering Jeopardy style challenges the steps of the solutions are always the same. Trading with the flags with other teams to achieve better position in the competition is a relevant risk in all CTF games. To prevent and deter plagiarism the CTF game provider has to monitor the solution steps to exclude teams with unrealistic solutions. Many CTF competitions have high prize rewards where great results can have professional benefit. To detect all plagiarism in such CTF games is essential nowadays.

Since each challenge is broken down into a series of attack steps, the challenge step dependencies can be transformed into indicators of compromise (IOC). These IOCs are artefacts of forensic evidence that are matched to logged events from the participant's interactions with the CTF challenges. Bianco (2014), [7] present the 'Pyramid of Pain' (PoP), as seen in Figure 1 that categorises types of IOCs that can be linked to a participant's activities. An example of how the PoP can be utilised with our challenge scenarios is the following: a CTF challenge step requires a user to submit a string of >31 characters to produce an error. The logged POST parameters are observable Network/Host Artefacts. These artefacts can then be matched other evidence such as IP Addresses, unique session IDs and web user-agents that compromised the CTF challenges to determine if the actions of a participant were legitimate or plagiarised. If a user compromises the challenge without fulfilling the steps then illegitimate activity has taken place.

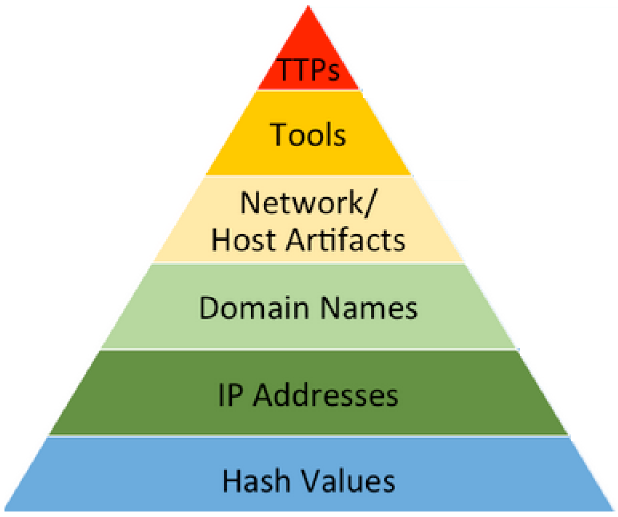


Figure 1: Pyramid of Pain - IOC types that can be used to detect a participant's activities [7]

The Hacking Arena environment hosted at the University of Oslo [4] is utilised to teach ethical hacking modules through a variety of web-based CTF style challenges. For this research we utilise HTTP logs gathered from two taught ethical hacking modules via the Hacking Arena's CTF challenges to aid and develop our research.

3. Related

A previous study on cheat detection in CTF challenges can be found in the work conducted by Kakouros (2020) [8]. This research utilises an inference engine and ad-hoc CTF infrastructure to capture and monitor the actions of players within challenges. However, this research is limited in that it does not take into account the sequence of events that took place, only matching steps independently and can therefore be manipulated by the user. Similarly, Kakouros' [8] approach is limited in its attribution of events to actors.

Previous studies have found that the utilisation of threat hunting methods with inference engines is effective at tracing the lifecycle of a threat actor's actions [9] [10] [11]. Al-Mohannadi *et al.* (2020) [9] utilise cyber threat hunting techniques with ELK [12] stack to analyse honeypot logs. Through keyword searching and visualisation tools provided by ELK, they were able to identify attack events from benign events, breakdown these attack events into various subcategories for further analysis. This research provides an insight into the effectiveness of inference engines like ELK stack for cyber threat hunting, however it is an analysis of all the honeypot log data rather than the attribution of events to an actor in a CTF environment. Similarly, relying on a manual iterative process to analyse the log data.

Al Shibana & Anupriya (2019) [10] propose an automated approach to threat hunting with inference engines. This research creates a series of detection rules tailored to indicators of compromise captured by Sysmon that are then indexed by the inference engine. When the forwarded events match the detection rule, the analyst is alerted. Whilst this approach is effective as an intrusion detection system, our research works inversely; we know how the challenge is to be compromised and must analyse past events and attribute these events to the participants who compromised the challenge. Similarly, [11] utilise indicators of compromise and an inference engine for assessing and classifying threat levels. This research however utilises Sysmon logs in windows clients rather than our scope that is focused on the logging of HTTP interactions gathered from CTF challenge clients. This presents opportunity to explore new methods of detection through the utilisation of a threat hunting methodology that translates a series of challenge dependencies into indicators of compromise (IOC), unique event signatures that are queriable by an inference engine. Whilst Daszczyszak *et al.* (2019) [13] indicate that IOCs are sensitive to polymorphism and metamorphism our web hacking challenges are static with expected attack patterns that a participant must fulfil. Therefore focusing on these IOCs is not problematic within our scenario.

4. Methods

Each web hacking challenge has a series of predefined dependencies that a participant should fulfill to acquire the challenge flag. This is because each step in the challenge provides further information or interaction to the participant to complete the challenge. By fulfilling each step within the challenge dependency, the detection system can determine the user as benign or, when steps are missing or out of order, the participant is deemed suspicious.

Every step of a challenge dependency is treated as an indicator of compromise (IOC) with unique elements that define them. This allows the detection system to query these specific elements, match the actions of participants to these IOCs across the entire index of captured data and analyse the series of steps for suspicious activities. Furthermore, we are the author's of the CTF challenges and therefore know what is required of a participant to get from one step to another. Because of this we can manually generate the IOCs that are to be searched for in the indexed security logs by the CTF querier [14]. Where steps can be fulfilled in multiple ways, it is possible to generate multiple IOCs for a challenge step and have the automated system query the series of possible IOCs to determine if a participant has fulfilled the challenge step.

Example steps can be found in Figure 2 for a brute web hacking challenge that requires a bruteforce attack to be conducted to acquire the challenge flag. To fulfill the dependencies the participant must first make a request to the web page to view its contents, interact with the web login form that is present on the site, conduct the bruteforcing attack, login to the site using the bruteforced credentials, acquire the flag. If a participant requests the flag file without conducting the brute force attack then this is suspicious activity.

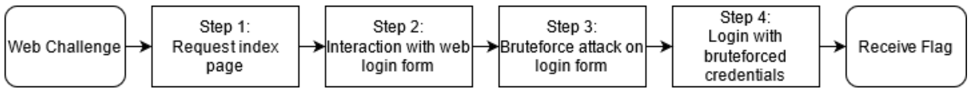

Web Challenge
Step 1: Request index page
Step 2: Interaction with web login form
Step 3: Bruteforce attack on login form
Step 4: Login with bruteforced credentials
Receive Flag

Figure 2: Example simplified steps required to fulfill a challenge dependency related to a web brute-forcing attack

4.1. Infrastructure

To be able to process and detect suspicious activities of participants within the web hacking challenges, the detection system requires access to the challenge logs.

To facilitate this, the infrastructure in Figure 3 has been developed to acquire the logs from individual web hacking challenges, process these logs into a universal format and then stores these logs in a centralised location for querying.

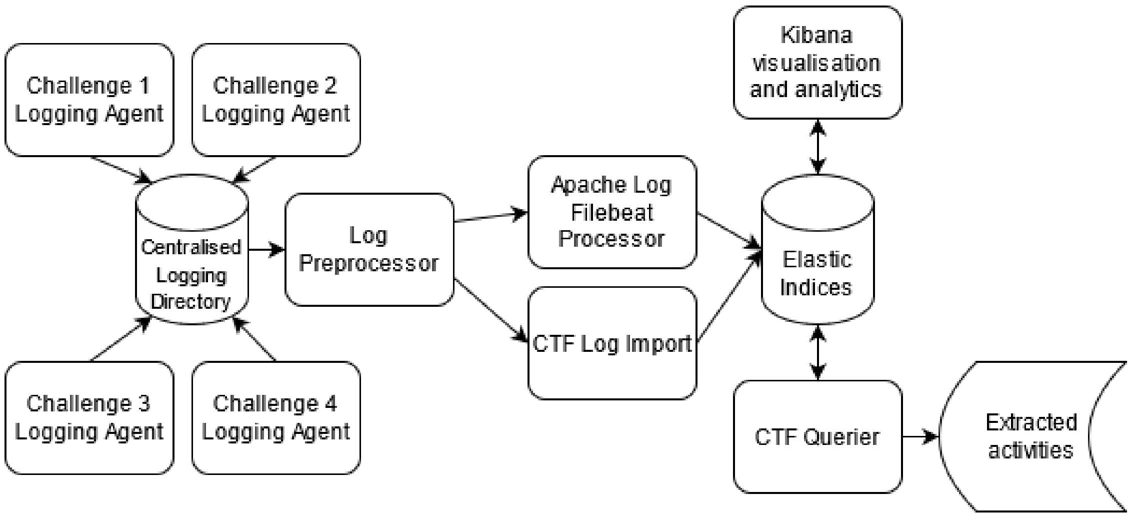

Challenge 1 Logging Agent
Challenge 2 Logging Agent
Centralised Logging Directory
Challenge 3 Logging Agent
Challenge 4 Logging Agent
Log Preprocessor
Apache Log Filebeat Processor
CTF Log Import
Kibana visualisation and analytics
Elastic Indices
CTF Querier
Extracted activities

Figure 3: Infrastructure design of detection system

To analyse each step and determine if an participant has fulfilled the dependencies, a threat hunting approach is applied. Each step can be transformed into an indicator of compromise (IOC) that is unique to each challenge.

Logging Agent: Each web hacking challenge has an agent that logs all HTTP interactions. These logs include all IP addresses, the timestamp of individual interactions, GET and POST requests, cookies and web user agents. Similarly, all Apache web server interactions are logged for each challenge. These logs are then analysed to match challenge dependencies to participant's activities. Focusing only on HTTP is sufficient enough as the current challenges are only web-based, where the challenges can only be interacted with via HTTP requests.

Centralised Logging Directory: To avoid overloading the infrastructure that facilitates the web hacking challenges, these logs are stored in a centralised directory located on a separate logging server. This allows for acquisition and processing of all logs from a single source point without the need to communicate with the web challenge agents. This solution is scalable as the only dependency is the capacity of the centralised system's storage space, where less or more storage space dictates the amount of challenges that can be logged.

Log Preprocessor: The log preprocessor then acquires all logs from the centralised logging directory. If the logs are not Apache access or Apache error logs then further preprocessing is required, otherwise the elastic Filebeat agent processes and parses the logs to the Elasticsearch Filebeat index. The preprocessing stage for non Apache logs converts the logs into a .csv formatted data set and applies additional logic to the logs dependent on the web hacking challenge category. Furthermore, these data sets can also be used for future research.

An example of this additional logic is the following scenario: The web hacking challenge requires a user to exploit a vulnerability in its search parameter. This vulnerability exposes a directory path when the user submits a search string of more than 20 characters. The preprocessing logic gets the length of each submitted search parameter and adds this to the dataset. Since this data isn't relevant to a challenge configured for SQL injection it is only processed for challenges that require such a vulnerability to be utilised.

Elasticsearch: After preprocessing, the data is output to Elasticsearch either as a Filebeat index containing all Apache access and error logs or as an index containing the custom CTF logs of a specific web hacking challenge.

Querying: The Elasticsearch indices can then be queried either through the automated CTF querier to get a holistic view of an participant's actions or by using the 'Discover' module in Kibana for search queries. The query parameters are defined by the web hacking challenge dependencies and look for matching signatures in the HTTP traffic. Limitations with Discover were found when querying multiple indices with different formatting types. Due to the content of the Apache logs and the Filebeat agent not being the same format, contents were missing when returning search results. This creates inefficiencies when using the Discover module for searching and detecting suspicious activities. Applying a field alias could be assigned to the field names in indexes as a workaround but this only returns the data of those field aliases. Therefore we have developed an automated CTF querying tool for interacting with Elasticsearch indices.

4.2. Logging

Each web hacking challenge is logged using two logging types. The first type is the Apache access and error logs that are generated by the Apache Web Server contained on each system. Access logs are formatted using the Apache combined log format to include the following elements:

- IP Address
- Timestamp
- HTTP request method (GET/POST)
- Status code
- Return byte size
- Referrer
- Web user agent

Error logs are formatted using the Apache default formatting and contain any errors returned by the system. These may include requested documents unavailable, PHP errors, access revocation.

The final logging type is a custom CTF logging format that is used for all web hacking challenges. This logging format contains the following elements:

- Timestamp
- IP address
- Challenge name
- Requested page
- HTTP GET content
- HTTP POST content
- Site cookies
- Web user agent
- Unique ID

By collecting and indexing these log elements they can be queried for indicators of compromise where the elements can be attributed to a participant fulfilling challenge dependencies.

4.3. CTF Querier

The CTF Querier is an automated tool that leverages the Python Elasticsearch Client [15] for querying the Elasticsearch indices. These queries are the IOCs that are generated from the challenge dependencies.

Since the CTF Querier is designed to work inversely, focusing on the steps conducted before the captured flag was obtained, the final flag query is used to obtain participants who compromised the web challenge and narrow the scope of the queries. This not only saves time and increases detection rates but ensures that the system only analyses the actions of subjects that compromised the web challenges. Once the system has obtained a list of participants that match the IOC of the final challenge step it conducts the following:

1. Get timestamp of final flag IOC for each participant.
2. For each step in a challenge dependency gather the following:
   a) Get all participants who match the IOC
   b) Get initial timestamp of IOC match for each participant
3. Check the fulfilment of challenge dependencies for each participant by checking the following:

   a) Did the participant complete the step or is a match missing? If missing then define the lack of action as suspicious.
      i. If so define the action as suspicious
      ii. Update suspicious list with participant details
   b) Did the challenge step occur before a previous step?
      i. If so define the action as suspicious
      ii. Update suspicious list with participant details
4. Create a report for the analyst that indicates the number of times an participant appeared in the suspicious list and the suspicious actions that are attributed to that participant

The following example provides a subset of IOCs for a CTF challenge which can be queried by the CTF querier. These logged HTTP requests indicate if a participant has fulfilled a step in the challenge dependency.

1. ”match_phrase”: {”query”: ”*audi”}
2. ”match_phrase”: {”query”: ”*\\/etc/passwd*”}
3. ”match_phrase”: {”url.original”: ”/index.php?car=php://filter/convert.base64-encode/resource=index.php”}
4. ”match_phrase”: {”url.original”: ”/loginforusers/index.php”}
5. ”match_phrase”: {”post”: ”POST: car=’ or position()=3]/*[5]|a[’;\”}

4.4. Testing of CTF Querier

For testing of the CTF Querier’s accuracy, several web hacking challenges were chosen with two simulated participant interactions with these challenges. The chosen web hacking challenges are of different challenge types to ensure the CTF querier isn’t suitable for only one challenge type.

The two participants always follow the following approaches:

- **Benign Actor:** This participant fulfills all challenge dependencies.
- **Malicious Actor:** For each challenge type the malicious participant does not fulfill challenge dependencies, deliberately misses challenge steps or completes steps out of order.

As the detection system predicts the participant’s set of actions as either benign or suspicious based upon matching the actions to IOCs, the following confusion matrix in Figure 4 can be used to quantify the performance of the detection system and compare the true positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) generated by the system. When the prediction for suspicious activities is correct this activity can then be matched to a TP. Similarly, correct predictions of benign activity is a TN. Incorrect predictions of suspicious activity are FP and incorrect predictions of benign activity are FN.

Precision and recall are derivatives of the confusion matrix used to statistically analyse the performance of the cheat detection system. Precision is the proportion of correctly classified positive predictions that belong to the positive class. An important metric for evaluation due to the impact of falsely reporting plagiarism for participants.

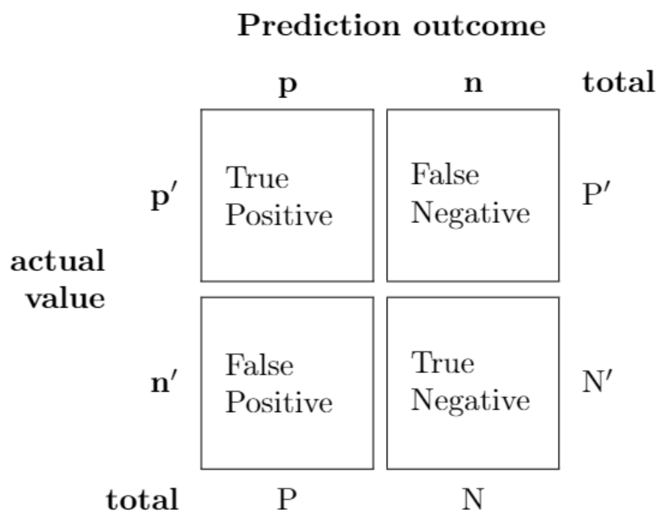


Figure 4: Confusion Matrix for cheat detection system prediction outcomes

$$Precision = \frac{TP}{TP + FP}$$

Recall is the proportion of events positively predicted within the whole dataset. Similarly to precision, there is a high impact to reporting plagiarism activities as false negatives.

$$Recall = \frac{TP}{TP + FN}$$

5. Results

To assess the performance of the detection system, four differing web hacking challenges were chosen for testing. Each challenge has a different series of challenge dependencies and overall goal for the participant to fulfill. The actions of each participant use the previously discussed benign and malicious participant approaches; where the benign participant conducts the series of steps in order whilst the malicious participant avoids steps.

The main purpose of the assessment is to determine the effectiveness in the detection system's ability to query the challenge dependency IOCs and define the suspicious or benign activities that are attributed to a participant.

For testing, the benign and malicious participant were simulated by the analysts. All results from these tests have been anonymised to preserve privacy.

The first web hacking challenge used for testing is an information disclosure challenge that requires the participant to submit a parameter string disclosed to the web server to retrieve the flag. The challenge dependencies state the following order of five actions:

1. The participant requests the index page of the challenge

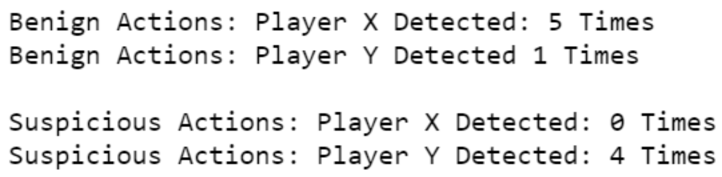
2. The participant requests and analyses the robots.txt, containing a list of directories to be excluded from web crawlers.
3. The participant requests the /Something/ directory.
4. The participant requests the /PennyLane/ directory.
5. The participant submits the web form with the string 'Hello' in the 'greeting' parameter. This returns the flag.

Our simulated set of actions for the benign user fulfill all of the challenge dependencies in order. However, these are the following set of actions for the malicious participant:

1. The participant skips all stages up to step 4. Actor fulfills step 4 by requesting the /PennyLane/ directory.
2. The participant fulfils step 5; submitting 'Hello' within the 'greeting' parameter.

Using the queries generated from the challenge dependencies for the parameter tampering challenge, the CTF querier analysed and defined all participant action's with 100% precision, 100% recall, 100% NPV and a 0% false positive rate, as seen in the Figure 6. confusion matrix.

Figure 5 provides an example output of the summarised events generated by the CTF querier. In this example, the results are obtained from the captured data related to the information disclosure challenge.


```
Benign Actions: Player X Detected: 5 Times
Benign Actions: Player Y Detected 1 Times

Suspicious Actions: Player X Detected: 0 Times
Suspicious Actions: Player Y Detected: 4 Times
```

Figure 5: Summary of classified set of simulated actions for information disclosure web hacking challenge

The next challenge for testing is a local file inclusion attack. The participant is required to inspect the source code of a server side script file by exploiting a local file inclusion vulnerability. The server side source code exposes a hidden site that is vulnerable to xpath injection attacks. By exploiting the xpath injection vulnerability the flag contained within an xml file is exposed. The dependencies of this challenge are as follows:

1. The participant requests the index page of the web challenge.
2. The participant analyses the website and interacts with the 'car' parameter that returns a .txt file prefixed with the brand of car the user inputs.
3. The participant tries out local file inclusion on the car parameter by requesting: **car=/etc/passwd/**
4. The participant tries to base64 encode the source file of the index.php page using the following string in the 'car' parameter: **php://filter/convert.base64-encode/resource=index.php**
5. Once the participant has decoded the base64 encoded source code they will see a reference to a **/loginforusers/"** directory. We expect the participant to request this directory.
6. The participant needs to conduct xpath injection on the login form to expose the flag using the following string: **' or position()=3]/*[5]|a['**.

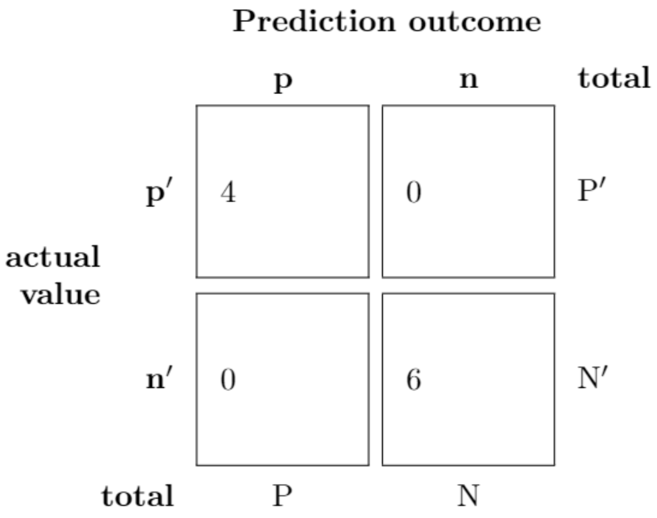


Figure 6: Confusion Matrix for cheat detection system prediction for information disclosure simulated test

a) Note: Whilst the participant needs to map the size of the XML document using the correct position numbers, if they use a randomised list of numbers to try it is probable they are successful on the first attempt.

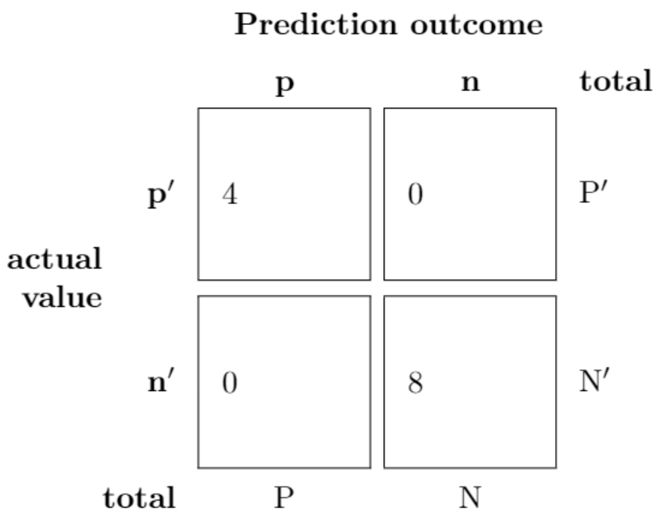
As before, the benign user conducts all steps in the challenge dependency. The set of actions for the malicious user are as follows:

1. Malicious participant requests the web challenge index page.
2. Malicious participant skips the local file inclusion steps. Skipping directly to step 5.
3. Malicious participant requests the '/loginforusers/' directory.
4. Malicious participant submits the correct xpath injection string to produce the challenge flag.

Using the queries generated from the challenge dependencies for the parameter tampering challenge, the CTF querier analysed and defined all participant action's with 100% precision, 100% recall, 100% NPV and a 0% false positive rate, as seen in the Figure 7. confusion matrix.

The third challenge used for the simulated set of actions requires the participant to post a string size greater than 31 characters to disclose a hidden directory in an error message. The set of actions are the following: the benign participant fulfills all challenge dependencies whilst the malicious participant is provided the flag URI by a friend, thus skipping all steps. The following confusion matrix in Figure 8 summarises the classified actions by the CTF querier.

The final challenged used to analyse the simulated set of actions requires the participant to tamper with a numerical query parameter until the correct parameter number is identified. The set of actions for this challenge is the following: the benign participant fulfills all challenge

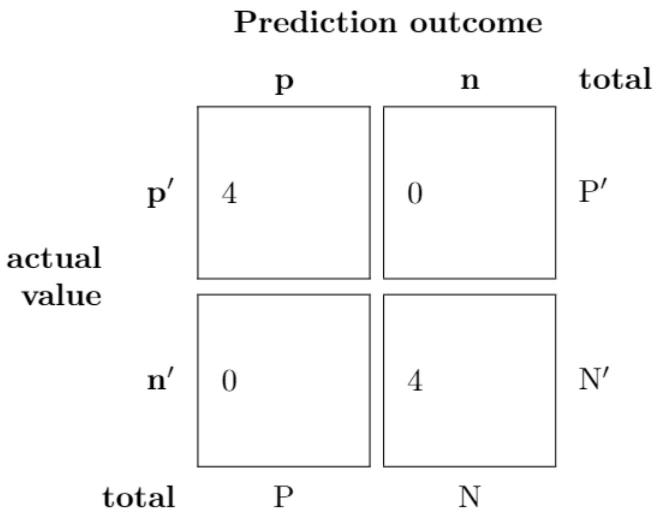


Prediction outcome

		p	n	total
actual value	p′	4	0	P′
	n′	0	8	N′
	total	P	N	

Figure 7: Confusion Matrix for cheat detection system prediction for file inclusion simulated test

dependencies, the malicious participant only requests the index page and submits the correct numerical parameter on first attempt. The confusion matrix in Figure 9 summarises the classified actions by the CTF querier.

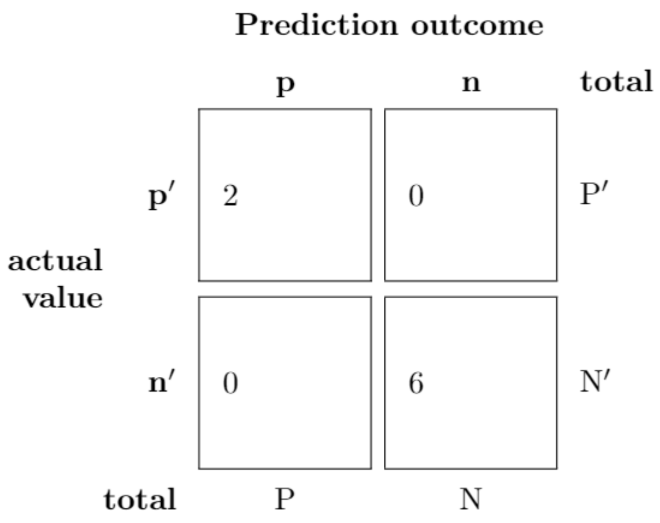


Prediction outcome

		p	n	total
actual value	p′	4	0	P′
	n′	0	4	N′
	total	P	N	

Figure 8: Confusion Matrix for cheat detection system prediction for third simulated scenario

The following confusion matrix in Figure 10 contains the values of all action classifications by the CTF querier for all challenges. The accuracy, precision and recall and NPV are all 100%

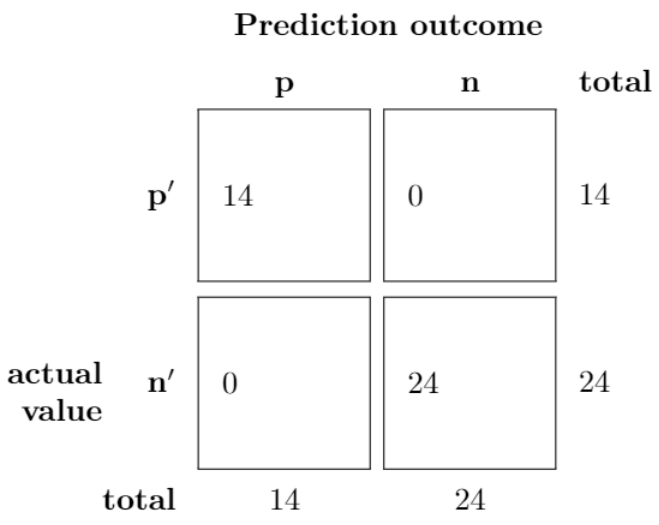


Prediction outcome

		p	n	total
actual value	p'	2	0	P'
	n'	0	6	N'
	total	P	N	

Figure 9: Confusion Matrix for cheat detection system prediction for final simulated test

with a 0% false positive rate when classifying the actions as either benign or malicious.



Prediction outcome

		p	n	total
	p'	14	0	14
actual value	n'	0	24	24
	total	14	24	

Figure 10: Final confusion matrix for all challenge predictions determined by the CTF Querier

6. Discussion

Our research goal is to explore the detection of plagiarism in CTF games. The CTF querier achieves this goal through the automated CTF querier. The application of a threat hunting method for generating signature based queries for challenge dependencies greatly increases the CTF querier's accuracy and in turn reduces the false positive rate when classifying a participant's actions as benign or suspicious.

Analysis of the final confusion matrix shows that the CTF querier is able to accurately perform multi-phase event detection and classification of captured web traffic for user-defined web hacking challenges. The CTF querier is capable of analysing vast quantities of indexed HTTP log data and correctly classify malicious and benign events and attribute those events to a participant with no false positives. Furthermore. by utilising signature based threat hunting methods to find IOCs pertaining to challenge dependencies, the CTF querier is not prone to false positives compared to similar research conducted by [8]. Kakouros' [8] approach to cheat detection in ethical hacking was prone to false positives in their preliminary testing due to the infrastructure failing to log events. This resulted in a sensitivity rate of 67% and an accuracy of 75% compared to the CTF querier that had no false positives and correctly classified each malicious and benign event. After Kakouros [8] reconfigured their infrastructure the accuracy was improved to 91%. There were no instances of events failing to log within the CTF querier infrastructure which further aids the confidence of our implementation. Our current implementation is limited to event attribution to individual participants and can not attribute group based plagiarism, in the case of participants sharing information to one another.

A limitation in this research is that is the current challenges are only web-based where a series of steps guide the participant to the challenge flag. Future work could expand into host-based CTF challenges where participants activities appear more unique from each other, requiring the system to detect and score IOCs automatically rather than manual entry into the CTF querier by an administrator.

Due to the scope of the CTF challenges being web applications the focus is only on the logging of HTTP traffic, therefore the CTF querier is currently limited to only analysing these types of events. However, the adopted threat hunting methodology can be adapted to analyse signatures in other log formats. Similarly, as the infrastructure is currently only using ELK stack the queries are made only for this vendor. Future work into generating queries from the challenge dependencies in a standardised format for SIEM tools could be achieved through the usage of the SIGMA rule formatting to achieve interoperability [16].

7. Conclusion

CTF style games are popular delivery methods for ethical hacking education; however there is limited research on verifying the legitimacy of CTF participant's activities and detecting plagiarism for educational examinations in CTF style games. To solve this problem, a lightweight automated querying tool called the 'CTF Querier' is proposed that queries participant's activities and checks for plagiarism or abnormalities in a fast, efficient and scalable way. This is achieved by combining cyber threat hunting methods with security analytics.

The combined method of cyber threat hunting and security analytics is achieved threefold. Firstly, by developing a simple, lightweight and interoperable HTTP logging format that is indexed in a centralised in a database for later querying. Secondly, transforming the steps a participant must fulfill into a series of IOCs for matching indexed participant activities and finally through the automated CTF querier that queries the challenge. By transforming the steps participants must take to fulfill a challenge into a series of IOCs, the presented CTF querier can automatically verify if a participant has fulfilled the challenge steps and match unexpected, missing or suspicious participant activities. Furthermore it is capable of performing this decision making without false positives.

For testing the accuracy and precision of the presented CTF querier, a dataset that contains the captured participant activities from several CTF style educational components alongside the captured activities of simulated participants is used. As the results show, the CTF querier can classify a participant's activities with high precision and no false positives when querying the dataset for the challenge step IOCs. Currently the presented CTF querier hunts for signatures in captured participant activities, future work for the CTF querier can include statistical methods to aid in machine learning based predictions of participants activities.

References

[1] HackTheBox, Hacking training for the best, 2021. URL: https://www.hackthebox.eu/.

[2] C. Academy, 2021. URL: https://ctfacademy.github.io/index.htm.

[3] M. Lehrfeld, P. Guest, Building an ethical hacking site for learning and student engagement, SoutheastCon 2016 (2016). doi:10.1109/secon.2016.7506746.

[4] L. Erdodi, Hacking arena security lab - department of informatics, nd. URL: https://www.hackingarena.no/home/index.html.

[5] J. Porup, Oscp cheating allegations a reminder to verify hacking skills when hiring, 2019. URL: https://www.csoonline.com/article/3336068/oscp-cheating-allegations-a-reminder-to-verify-hacking-skills-when-hiring.html.

[6] J. Adams, The dangers of exam dumps, 2016. URL: https://www.cbtnuggets.com/blog/career/career-progression/the-dangers-of-exam-dumps.

[7] D. J. Bianco, The pyramid of pain, 2014. URL: http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html.

[8] N. Kakouros, A cheat detection system for an educational pentesting cyber range: an intrusion deficit approach, Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2020.

[9] H. Al-Mohannadi, I. Awan, J. Al Hamar, Analysis of adversary activities using cloud-based web services to enhance cyber threat intelligence, Service Oriented Computing and Applications (2020). doi:10.1007/s11761-019-00285-7.

[10] M. Al Shibani, E. Anupriya, Automated threat hunting using ELK stack – a case study, Indian Journal of Computer Science and Engineering (2019). doi:10.21817/indjcse/2019/v10i5/191005008.

[11] V. Mavroeidis, A. Jøsang, Data-driven threat hunting using sysmon, in: ACM International Conference Proceeding Series, 2018. doi:10.1145/3199478.3199490.

[12] Elastic, Elk stack: Elasticsearch, logstash, kibana, n.d. URL: https://www.elastic.co/what-is/elk-stack.

[13] R. Daszczyszak, D. Ellis, S. Luke, S. Whitley, 2019. URL: https://www.mitre.org/sites/default/files/publications/pr-19-3892-ttp-based-hunting.pdf.

[14] R. A. Chetwyn, Ctf querier, 2021. URL: https://github.com/chetwynr/CTF-PlagiariasmDetection/.

[15] S. M. Larson, Python elasticsearch client, 2021. URL: https://elasticsearch-py.readthedocs.io/en/v7.12.1/.

[16] F. Roth, T. Patzke, Sigma - generic format for siem systems, ???? URL: https://github.com/SigmaHQ/sigma.