National College of Ireland

Automated Threat Hunting for JavaScript-based Obfuscated Phishing Email Attachments

MSc Industrial Internship
MSc Cyber Security

Saraunsh Shewale
Student ID: X21215057

School of Computing
National College of Ireland

Supervisor:	Vikas Sahni
Industry Mentor:	Colm Gallagher

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Saraunsh Shewale
Student ID:	X21215057
Programme:	MSc Cyber Security
Year:	2023-2024
Module:	MSc Industrial Internship
Supervisor:	Vikas Sahni
Submission Due Date:	05/01/2024
Project Title:	Automated Threat Hunting for JavaScript-based Obfuscated Phishing Email Attachments
Word Count:	5847
Page Count:	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Saraunsh Shewale
Date:	29th January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Automated Threat Hunting for JavaScript-based Obfuscated Phishing Email Attachments

Saraunsh Shewale
X21215057

Abstract

Phishing is a well-known social engineering attack vector that targets employees and high-level executives to trick them into disclosing their user account credentials. Emerging phishing techniques employ the use of obfuscated JavaScript code within *.html* file attachments. Such techniques bypass most of the advanced security protections in place. This research presents a lightweight, easy-to-set-up automated threat intelligence workflow that is focused on the extraction of potential Indicators of Compromise (IOCs) from suspicious emails. It helps to uncover and flag suspicious artifacts from the email and its attachments including IP address, email address, file hash, and URL. The project is built over a cloud-based SaaS service - Tines, and it leverages the effectiveness of existing open-source and commercial security services like VirusTotal, URLscan.io, EmailRep.io, and OpenAi.

1 Introduction

1.1 Research Background

Businesses frequently suffer from cyber security breaches despite employing robust security solutions such as endpoint detection and response (EDR), anti-malware software, next-generation firewalls (NGFW), and whatnot. This is because the weakest link in the whole cyber security chain is a human! Social engineering attacks are weaponized in such a way that these modern security solutions fail to distinguish between benign and malicious code sent from the outside world over corporate email channels. This leads to successful email delivery and recipients getting greeted with phishing emails in their mailboxes. When a user clicks on the link received in a phishing email, they will be redirected to some sort of login page from known product vendors for example Microsoft Office login, Google account login, or any other social media and networking sites login page to trick the users into obtaining potential personally identifiable information (PII). Managed security service providers (MSSPs) with 24*7 security monitoring teams unable to individually inspect each inbound email and its content passing through corporate mail channels as they're already dealing with a lot of alert fatigue. There are no such online services that specifically automate the threat intelligence workflow for suspicious emails attached with obfuscated JavaScript code which serves as a phishing link redirector and thus the need for such an automated system stands.

As per the 2023 Verizon Data Breach Investigation Report (DBIR)[1], a total of 74% of data breaches were held accounted by humans due to the error they made, and a majority

[1]Verizon DBIR: `https://www.verizon.com/business/resources/reports/dbir`

of these breaches fall into the category of social engineering attacks. The most widely seen and frequent of current times is known as Phishing. Gaining end user's trust and obtaining sensitive information such as usernames, passwords, email addresses, credit card details and other personally identifiable information (PII) is the primary goal of a phishing attack. This may include targeting a single employee or an entire enterprise and its high-level executives to trick them into divulging their sensitive information to conduct illicit operations.

From an enterprise context, an attacker would craft a malicious phishing link and forward it to mass audiences or enterprise employees in the event of a targeted phishing attack where end-users fail to distinguish the email's legitimacy and end up giving their personal information. In a traditional phishing attack scenario, an attacker would embed a phishing link directly in the email body which can be easily detected by modern anti-malware solutions and email gateways. This leads to direct quarantine of the phishing email and thus blocking the phishing attack in the first place. However, modern phishing attacks work in a more stealthy way where an attacker builds malicious JavaScript code with a redirector functionality that redirects the end user to a phishing URL. The crafted code is then obfuscated using JavaScript obfuscators and packers such as Obfuscator.io[2] and then attached to the email as a .HTML file. Obfuscation adds redundant data to the original code to hide the logic of the code and makes the code irrelevant. It is based on the principle of obscurity. The obfuscation helps to evade and bypass detections from modern anti-malware solutions which results in email being dropped into the user's mail inbox without getting caught. This bypasses the first line of defence in the security workflow and doubles the possibility of a successful phishing attack. Currently, modern security tools cannot evaluate/de-obfuscate the obfuscated code at runtime. Thus, attackers leverage these techniques to bypass the security solutions.

In this area of research, an automated solution/workflow is designed to extract potential indicators of compromise (IOC) from suspicious emails, and their attachments even if the attached files are obfuscated using JavaScript packers. The automated workflow can save the time spent by security analysts on suspicious email analysis and helps to minimize the mean-time-to-detect (MTTD) metric. Existing tools and online threat intelligence services do not specifically address the problem of automated IOC extraction from suspicious emails in the threat intelligence area. However, some tools are capable of identifying and analysing the obfuscated JavaScript code, but it lacks the threat intelligence perspective and does not help to identify and flag potential indicators of compromise.

1.2 Problem Definition

The frequency of cyber-attacks is increasing at an alarming rate and has been ranked under the top ten threats in the year 2020 in terms of likelihood and impact. Among many cyber threats, one being the most prevalent is social engineering attacks. In which people are tricked into disclosing their confidential information including but not limited to usernames, passwords, credit card details, social security numbers, and the list goes on. Attackers have discovered new ways to trick the users using obfuscated JS code embedded into .HTML file which is then attached to a phishing email. Once the file attachment is opened by the victim, obfuscated JavaScript code redirects the user to a

[2]Obfuscator.io: `https://obfuscator.io`

phishing website. Such types of emails cannot be detected by security tools as obfuscation bypasses all existing security controls. Security analysts and incident response teams cannot individually inspect every suspicious email manually as it's a waste of effort. The process should be streamlined and made efficient to identify suspicious artifacts from the email in an automated fashion to save time and maximize the detection rate of indicators of compromise (IOCs).

Research Question – How can the threat-hunting process be automated for a suspicious email and associated obfuscated JavaScript attachments to extract potential indicators of compromise (IOCs)?

The project increases the security team's efficiency by a relatively significant percentage in terms of time spent in the investigation phase for a suspicious email and its obfuscated file attachments. The automated workflow will incorporate various open-source and commercial API services for static analysis which will help to extract associated indicators of compromise (IOCs) and assign them a credibility score.

This research paper details and discusses work related to the detection of obfuscated code and static/dynamic analysis models for malicious code detection in section 2. The employed research methodology for the project is discussed in section 3. Design details and implementation specifics are outlined in section 4 and 5 respectively. The project outcomes are evaluated and discussed in section 6 followed by a discussion on the conclusion and future work in section 7 of this report.

2 Related Work

This section discusses recent and past literature research on three major areas in the scope of the presented area of research. It is based on the detection of malicious obfuscated code written in JavaScript and static analysis on the suspicious code followed by querying real-time analysis results from multiple threat intelligence services to associate a reputation score for each of the extracted indicators-of-compromise (IOCs).

2.1 Detection of Obfuscated Code

JavaScript obfuscation has proven to be a very powerful security bypass technique to evade modern anti-malware detection. This has been presented by Xu et al. (2012) in a measurement study on obfuscated techniques for malicious JS code. More than 20 anti-malware solutions were used as part of this study to determine the stealth factor of obfuscated JS code.

Identifying patterns of obfuscated code from source code is one of the key steps in code classification. The code classification is based on techniques such as tokenization and predictive analysis. A novel methodology proposed by Choi et al. (2009) discussed suspicious obfuscated code detection using static string pattern analysis in web pages. The method used for code analysis in this research operates on three (3) parameters which are N-gram, entropy, and word size to detect malicious patterns in JavaScript code. JavaScript functions such as eval(), document.write() and other dangerous function calls can be detected using the method proposed in this research. However, the proposed method only supports the detection of four (4) malicious patterns out of the proposed

six (6) predefined patterns. Due to the minimal number of predefined patterns, it limits the detection capabilities of suspicious code. Morishige et al. (2017) proposed a novel approach to detect and extract hostile URLs from obfuscated code using a machine learning model that uses frequency analysis techniques. To avoid detection by anti-malware programs, modern obfuscators employ a divide-by URL method. This method divides part of the URL into multiple separate variables using the '+' operator and then it's reconstructed at a later stage in the code. The proposed method in this research allows the detection of this type of obfuscation pattern using predefined three (3) methods. The resultant outcome demonstrated a reduction in false negative rate by 20% and it was found ineffective when the URL was reconstructed using different obfuscation methods other than the predefined methods.

The primary objective of an obfuscator is to maintain the original logic of the code while salting it with unnecessary data. This makes the code unreadable and bigger in the aspect of code lines (LoC). Talukder et al. (2019) proposed research that takes obfuscated code as an input and slices it into parts for easy interpretation. It's possible to break down large, obfuscated code into multiple parts for manual inspection. However, it lacks automatic analysis and code evaluation support. A similar detection method was proposed by Lu and Debray (2012) which is based on Semantics analysis to automatically de-obfuscate the code and replicate the logic of the original code. Code is executed dynamically inside the web browser to determine the trace of the next byte code going to get executed. This requires the additional overhead of provisioning a safe isolated environment for executing the code in a web browser.

2.2 Static Analysis for Obfuscated Malicious Code

Static analysis involves a process by which a code is analysed without executing it at runtime. The primary focus in this testing strategy involves the inspection of source code and comparing that with a malicious code sample.

Srndic and Laskov (2013) proposed a static detection methodology to detect the presence of malicious JavaScript code injected inside PDF files. The research is based on a tokenization technique to extract code samples followed by its analysis on threat intelligence platforms such as VirusTotal. The research published an open-source tool known as PJScan. However, the tool has a higher false positive rate and it's less reliable as a threat intelligence solution. Another limitation of this open-source project is that it only works on PDF files and does not serve its purpose for other file types. Likarish et al. (2009) discussed the use of classification techniques to detect obfuscated malicious JavaScript code based on feature extraction mechanism using machine learning classifiers. According to the authors, this classification technique might flag legit obfuscated code as malicious if packed with any known JavaScript packer.

Another relatively recent research on malicious JS code detection based on LSTM (Long Short-Term Memory) was published by Fang et al. (2018). The authors demonstrated feature extraction from the semantic level of byte code which resulted in malicious JS code detection with an accuracy of 99.51%. However, the research is limited to the detection of malicious code and does not solve a code evaluation problem. One very good research based on hybrid analysis using a machine learning attack model to detect and classify malicious JavaScript malware was published by Wang et al. (2015). The proposed tool JSDC successfully detects malware using predictive features of program structures and risky function calls. JSDC performed well in the detection of malicious

JavaScript programs and resulted in a very low false positive rate of 0.2123

Detecting malicious drive-by-download campaigns through dynamic crawling and execution methods may become prohibitive in most cases. To tackle this problem, a novel static analysis approach (JSPRE) was proposed by Hou et al. (2018) using a malicious page collection algorithm to classify malicious campaigns using guided crawling. JSPRE was able to detect malicious web pages written in JavaScript with higher accuracy but is only limited to the identification of illicit code. A similar problem of drive-by-download attacks was addressed by Cova et al. (2010). The authors proposed a novel machine learning approach where obfuscated code is detected by emulating its behavior followed by comparison with benign code profiles. Relatively new research for detecting obfuscated JavaScript code was proposed by Dujmović et al. (2023). The proposed tool in this study first crawls web pages and their content with the help of a request library in Python and is later analysed based on 6 (six) parameters – regex, entropy, size of the biggest word, average word size, file size, and the ratio of largest word to count of characters. The tool further categorizes analysed content into three (3) different categories – source code, obfuscated code, and minified code. The study identified the source code with an accuracy of 45%, minified code with 53%, and 2% code identified as an obfuscated code. The confidence score for obfuscated code was found to be relatively low and cannot be relied upon to build a reliable statistical analysis model. However, this research demonstrates the efficient use of regex (regular expression) to detect obfuscated patterns from the source code and it is found to be resourceful for the presented research.

2.3 Dynamic Analysis for Obfuscated Malicious Code

Dynamic analysis testing tests the application code at runtime i.e., the code first gets executed and analysed for its behavior. When the code is executed, its associated function calls and arguments invoked by the application are analysed to figure out its intent.

A fairly recent study from Starov et al. (2019) made a huge contribution to malicious coin mining campaigns which are coded in JavaScript language and later obfuscated to bypass anti-malware protections. The study focused on a dynamic analysis approach and helped determine runtime alert popups, global variables usage, web socket connections, and malicious API calls. Out of 9,104 thousand coin mining scripts, 4,788 thousand scripts were flagged as malicious scripts based on 250 plus malware signature detections from threat intelligence services such as VirusTotal. Based on code instrumentation, new research on JavaScript obfuscation detection using hybrid analysis was presented by He et al. (2018). This method incorporated static and dynamic analysis methodologies in a web browser plugin called MJDetector which is capable of performing obfuscated JS detection and evaluation at runtime i.e., while browsing web pages. The browser plugin resulted in high accuracy with 94.76% in obfuscated code detection and 100% accuracy with de-obfuscation of the same code for specific types.

Another similar hybrid analysis model incorporating static and dynamic analysis checks for obfuscated JS code detection was proposed by Kılıç and Sandıkkaya (2023). The proposed method in this study uses a machine learning model based on AST-based syntactic and lexical analysis to detect suspicious code patterns in obfuscated code. The results yielded in more accurate detection of such patterns and a lower false positive rate. However, the method has some limitations due to predefined patterns and can be less reliable if the code is obfuscated with new techniques. The approach taken by Starov et al. (2019) contributed as a good foundation for the research proposed by Galdi et al.

(2022). The authors made a great contribution to this area of research by proposing an open-source project known as ThePhish. It is capable of automating the entire analysis workflow starting from the extraction of observables from the email headers, body and file attachments and later scoring them based on their reputation score. The proposed framework leverages various APIs and open-source threat intelligence services including MISP, TheHive and Cortex. Based on the email characteristics, the tool generates a confidence score and flags it as malicious or non-malicious. In the case of a low confidence score, it will flag it as suspicious for manual intervention by a security analyst. However, it does not have a module to extract potential indicators of compromise from obfuscated JS code and thus leaves us scope for further addition on top of it. Also, this framework requires multiple dependencies for successful installation which includes setting up an email address, a Linux host machine with a specific Python version, and fully functional instances of TheHive, MISP, and Cortex.

Based on the gap identified concerning dependencies overhead, the presented research project allows running automated phishing analysis workflows with very few dependencies and it comes with a lightweight architecture compared to the ThePhish project.

3 Methodology

The approach taken for this project originated from the research onion (refer figure 1) concept proposed by Saunders et al. (2019). The research onion helps to lay down the research methodology holistically while thoroughly covering all aspects and objectives.

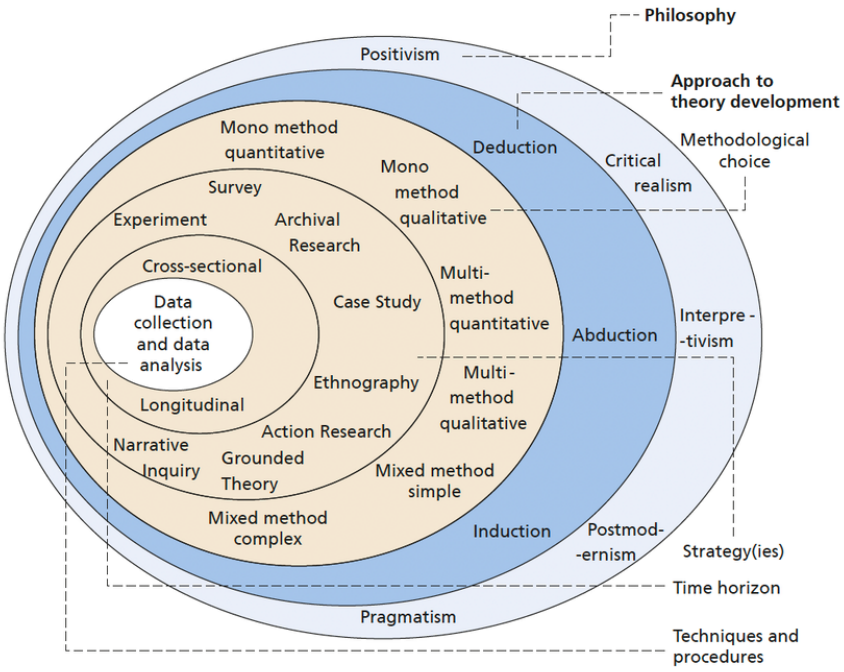

Figure 1: The Research Onion - Saunders et al. (2019)

The choices of research methods for this research were based on the research philosophy of positivism. Positivism is based on the measurability, objectivity, and use of quantitative data. The presented research in this area aligns well with the positivism

philosophy as it's centered on empirical analysis of observations. Also, it's based on the principle that knowledge can only be true, false, or meaningless. If the analysis cannot be concluded by either true or false, then that analysis cannot hold the ground anymore and can be dismissed.

The inductive research approach seemed an ideal choice for the presented research as it involves in-depth static analysis of obfuscated JS code related to real-time phishing incidents.

The mono-method qualitative process was considered for the research as obfuscated phishing file attachments involve analysis of non-numeric, textual, and random patterns of string data.

Concerning research strategy, a case study approach was selected due to the fact of a small number of phishing email cases for analysis. A case study approach drills down into specifics and it is suitable for a smaller number of cases. This helped to narrow down the in-depth analysis of certain phishing cases for a holistic exploration of outcomes.

The research choices influenced the consideration of Tines SaaS service platform along with multiple open-source and commercial threat intelligence services due to their less complicated architecture.

In accordance with time horizons, a longitudinal time horizon method was employed as the research is based on developments in the threat intelligence area over a prolonged period. This allowed the research methodology to track new changes and monitor the advancements made over time in the presented research area.

The methodologies and project planning techniques used in this research were influenced by an incremental model in the software development life cycle (SDLC). In the incremental process model, the project is divided into multiple smaller increments or parts and every new increment has additional changes on top of previous increments. The model is well depicted by Pressman (2010) and is mentioned in the figure 2.

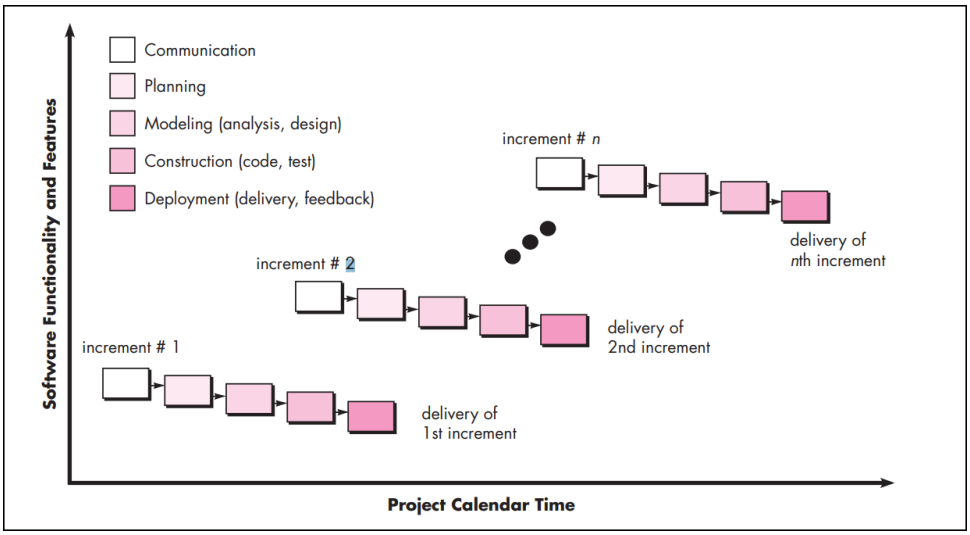

Figure 2: Incremental Model in SDLC - Pressman (2010)

The model walkthroughs various stages in SDLC such as communication, planning, analysis model and design, code building, and reporting.

3.1 Communication

The first and foremost step in the presented research methodology was communication where requirements were gathered from industry mentors on what had to be achieved. Detailed steps were outlined in the discussion phase on the development of an automated phishing analysis workflow for analysing suspicious emails flowing through the client's email gateway. The plan to develop such a workflow arose due to the encounter of a unique phishing incident where an email was attached with an obfuscated JS code. This email was not flagged by Office ATP (advanced threat protection) service due to the usage of a stealthy obfuscation technique. In this phase, the objectives and next action items were discussed to build a lightweight yet powerful suspicious email analysis workflow.

3.2 Planning

The required tools and online API services for the research project were decided in the planning phase. The objective was to build a workflow cost-effectively with low dependency overhead. Upon holistic consideration, a decision was made to opt for a cloud-based SaaS automation framework – Tines. Tines is a cloud service that allows individuals and security teams to build security orchestration workflows where everything can be automated. After platform selection, API keys for open-source and commercial API services were provisioned. The selection included services such as VirusTotal, URLScan, EmailRep and OpenAi due to their pricing factors and documentation support.

3.3 Analysis Model & Design

In this research, a hybrid static analysis model was considered for analysing suspicious emails while integrating new and past advancements in relation to obfuscated pattern detection and IOC extraction. The system design was restricted to the Tines platform which is a cloud-based SaaS service due to platform offerings and no installation overhead. The free subscription on this platform allows users to host three (3) workflows for free while the commercial plan allows multiple workflows with additional benefits.

3.4 Code Building

An incremental approach was taken and due to that, the project was divided into multiple smaller components to effectively build and test every aspect of functionality. Following is the list of components.

3.4.1 Email Parser

This component is used to parse the received email content in appropriate categories in JSON notation. JSON notation is based on the key-value pair principle, and it makes the programmatic job easy due to effective and easy content parsing. A dedicated email address is automatically provisioned by Tines for receiving emails for analysis. Once the email is forwarded to the automated workflow, this component kicks in and categorizes various sections of the email into different key-value pairs such as email headers, body, attachments, etc.

3.4.2 IOC Extraction

After successfully parsing email content, indicators of compromise (IOCs) are extracted which include IP addresses, web links, file hashes, emails, and other suspicious strings. This component is integrated with strong pattern-matching regular expressions (regex) which can detect and extract all IOC categories.

3.4.3 File Analysis

The file analysis component initiates static analysis checks if the parsed suspicious email consists of any file attachments. This component is one of the primary components that is responsible for detecting obfuscated JavaScript code patterns followed by IOC extraction. This component is iterated over OpenAi's function block which helps in de-obfuscating the detected obfuscated code for better understanding of the code. Also, the file signatures are calculated and passed to the querying model in the automated workflow.

3.4.4 Querying Threat Intel Services

This is the primary component in the automated workflow which is responsible for querying online threat intelligence services to determine reputation score for each extracted IOC. This component queries services such as VirusTotal, URLScan, and EmailRep to determine IP, URL, and email reputation scores respectively. This further calls OpenAi's API service in the event of obfuscated code detection for code simplification. The simplified obfuscated code is again iterated over the file analysis component for IOC extraction.

3.5 Reporting

This is the final step in the automated workflow which is responsible for consolidation and formation of overall analysis results. It makes use of hypertext markup language or HTML to format the results in email email-compatible body. The analysed results are categorized based on different IOC categories along with their determined reputation score which states if the IOC is malicious or not. The results are shared to the same email address from which the suspicious email was forwarded for analysis. Analysis outcomes can be also viewed in the Tines platform if the user has read access to the developed automated workflow.

4 Design Specification

A cloud-based SaaS platform- Tines[3] was used to automate the threat-hunting process for suspicious emails. Tines is a smart automated workflow builder which allows users to build automated analysis workflows with no additional setup and installation overhead. The major overhead of setting up a dedicated mail server for sending and receiving emails was solved by the Tines self-deployed mailbox cluster. This offered a unique mailbox and webhook address to forward suspicious emails for analysis. The code is written using pre-built and customized event block items provided by the platform. The predefined functions work on JSON notation to easily access JSON key values. Actions are core

[3] `https://tines.com`

components of the Tines platform where the code logic is defined. It supports (7) such action types which are mentioned in Table 1.

Table 1: Tines Supported Action Types

Action Type	**Description**
Send Email	Sends email to specified recipients
Event Transformation	Contains different modes to operate on data such as implode, explode, de-duplicate, etc
HTTP Request	Sends HTTP request for API calls
Receive Email	IMAP action for receiving forwarded emails
Trigger	Compares the content field with predefined rules (if, else)
Webhook	Receives HTTP callbacks
Send to Story	Transfers the control flow to sub-story within Tines

The system design of automated phishing analysis involved setting up API keys for multiple threat intelligence services which are mentioned in the project configuration Table 2 along with required dependencies for successful project execution.

The VirusTotal[4] API was used to determine the IP reputation score and URLScan[5] for the website reputation score. These two services were chosen due to their free API interface and the amount of threat intel feeds. A relatively recent paper by Masri and Aldwairi (2017) mentions the use of these services for the detection of malicious advertisement campaigns.

Similarly, EmailRep[6] API was used for checking email address credibility scores, and OpenAi[7] API for simplifying the obfuscated JS code. The idea of de-obfuscating obfuscated JavaScript code using OpenAi's API interface was influenced by the latest financial chatbot service - Kira proposed by Búadóttir et al. (2023).

Table 2: Project Configuration

Dependency Type	**Value**
Platform (SaaS)	Tines Account
API Keys	VirusTotal URLScan.io EmailRep.io OpenAi API
Web Browser	Chrome (Version Used - 120.0.6099.130) OR Firefox (Version Used - 121.0)

The receive email action type was used to receive emails forwarded for analysis followed by setting up event triggers. Triggers helped to identify and match each IOC category from the parsed email content such as IP address, URL, email, and files. To match

[4] `https://www.virustotal.com`
[5] `https://urlscan.io`
[6] `https://emailrep.io`
[7] `https://platform.openai.com`

these categories, regular expressions were integrated into the trigger events. Matched trigger conditions were then passed to subsequent event transformation actions to extract and explode each IOC to its subsequent HTTP request actions.

These actions query multiple threat intelligence services to determine the reputation score associated with it. Once the response is fetched, the results are formatted using event transform actions and later consolidated in a single event using implode event transform action. The imploded results are then forwarded to send email action for sending the results to a suspicious email forwarder i.e., the analysis initiator.

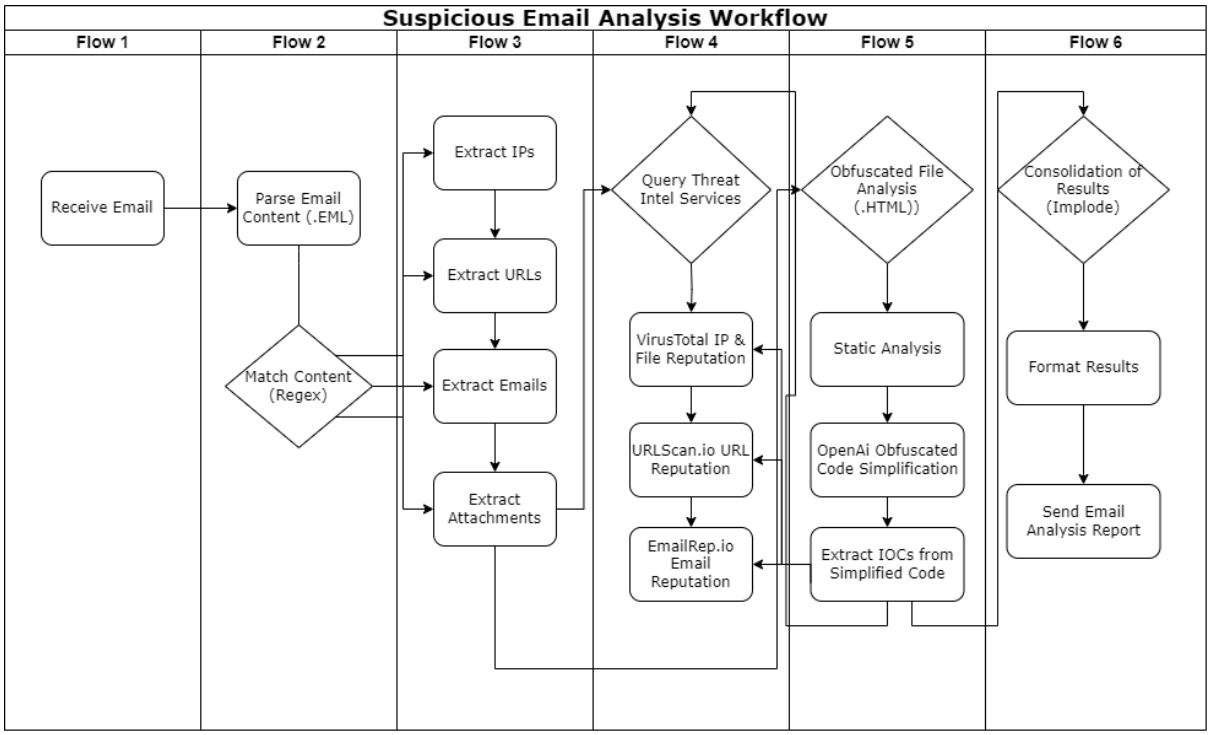

Figure 3: Project Flow Chart

Since the analysis is performed on the cloud, there is no need for high-end host system configuration except for the requirement of a reliable internet connection to check the event's status of the analysis.

5 Implementation

The presented project is built over a cloud-based SaaS service - Tines. The other primary building blocks of this project are based on open-source and commercial API services provided by VirusTotal, URLScan.io, EmailRep.io, and OpenAi respectively. Detailed implementation details are discussed in the following subsequent sections.

5.1 Provisioning API Keys

The automated workflow is backed by its reliable static analysis techniques and more importantly integrated threat intelligence API services. API interfaces are used to determine reputation scores of IP addresses, website links, and email addresses. These API

services require authorization keys to query the results from their database. For the same purpose, API keys for all of these services are integrated into the Tines platform's credentials section.

5.2 Building Workflow

Automated phishing analysis workflow for suspicious emails was built using Tines (SaaS service) inbuilt action types while customizing the options for each event transform logic. Receive Email and Webhook actions were dragged onto the storyboard of the platform. Tines automatically provisioned an email address for the story with *@tines.email* domain and the same for the webhook address.

An event transform action was then placed for parsing the email content into different key-value pairs based on JSON notation. The output of the email parser is plugged into event triggers which are used to match certain conditions and based on that it will pass the control to subsequent event handlers. Triggers were set up to detect the presence of any IP, URL, or Email in the parsed email body. For this purpose, regex was used. The table 3 defined various regex used for event triggers.

Table 3: Regex for Parsing Email Content

Use case	**Regex**
Match Email	`\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}\b`
Match IP	`\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b`
Match URL	`[A-Za-z]+:\/\/[A-Za-z0-9\-_]+\.[A-Za-z0-9\-_:%&;\?\#\/.=]+`

In the event of multiple such artifacts, the events are exploded to make iterative calls to threat intel services. The HTTP request action was used to query the threat intel service for determining the reputation score associated with each category of IOC. Subsequent calls are made to VirusTotal, URLScan, and EmailRep to gather reputation scores. A delay event transform is added to wait for a couple of seconds before requesting another resource. A sample of VirusTotal API request and response is shown in (figure 4) and (figure 5) respectively which showcases the response return with a safe reputation score.

Figure 4: VirusTotal Request for IP Reputation

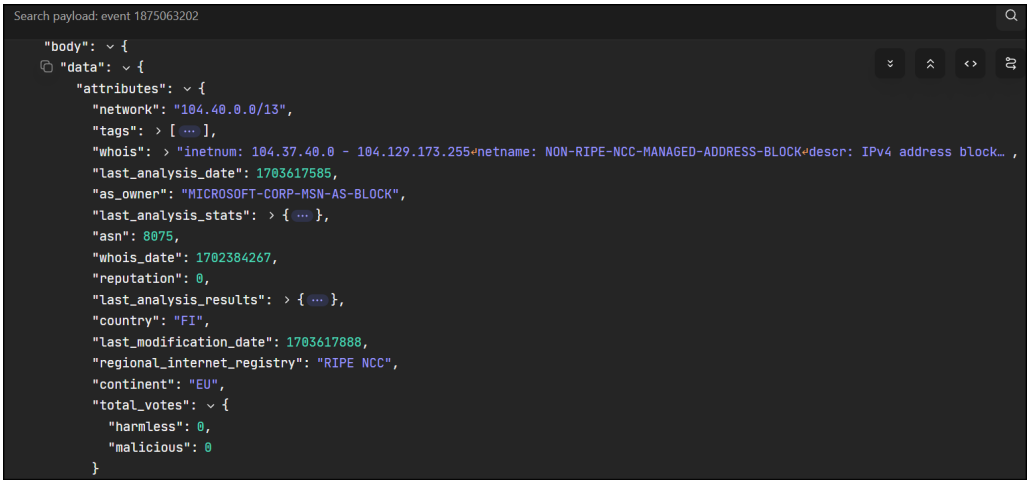

Figure 5: VirusTotal Response for IP Reputation

If there are files associated with an email, the trigger passes the control to the file analysis event transform where the file content is parsed. The parsed content is then analysed using static analysis techniques for the detection of any obfuscated code. Upon encounter of obfuscated code, the packed code is simplified by querying OpenAi's API interface. The simplified code is again iterated over IOC extraction event transforms and extracted IOCs are checked for reputation score. The obfuscated code submission form for analysis was created as depicted in (figure 6) for manually pasting the suspicious obfuscated code.

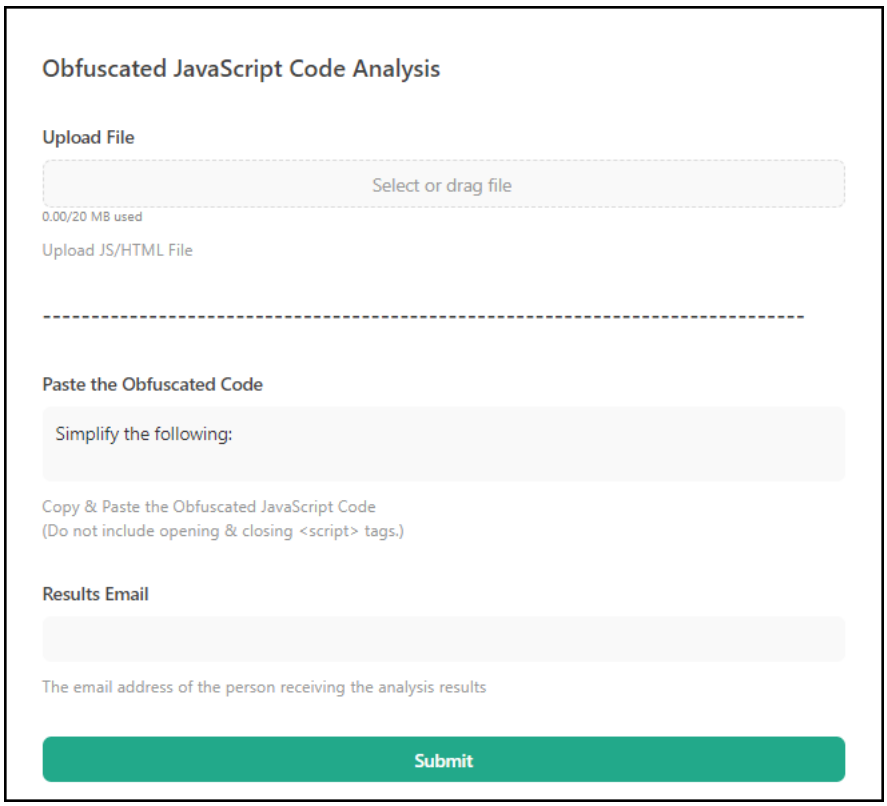

Figure 6: Code Submission Form

5.3 Result Formation

Once the loop iterates over all extracted artifacts, the results of the analysis are formatted using the build results event transform action where every piece of analysis is consolidated. The consolidated results are formatted in tabular form using hypertext markup language or HTML. The results are then sent to the same email address from which the email was received for analysis using the send email action in Tines. The code shown in (figure 7) is responsible for result formation.

```
{
  "recipients": [
    "saraunsh.shewale@commsec.ie"
  ],
  "sender_name": "Tines Security Automation",
  "subject": "Suspicious Email Analysis Results - <<DATE(parse_eml.email.date, \"%d %b, %Y\")>>",
  "body": "Hello Team,\n<br><br>\nThanks for sending the email to suspicious mail box. See the attached results from the
analysis.\n<br><br>\n<b>Please note:</b> Zero (0) reputation score means the IOC is clean/safe.\n<br><br>\n<style>\ntable \n{\n
border:1px solid #b3adad;\n    border-collapse:collapse;\n}\n    table th {\n    border:1px solid #b3adad;\n    padding:10px;\n
background: #fceec2;\n    color: #313030;\n}\n    table td {\n    border:1px solid #b3adad;\n    text-align:left;\n
padding:6px;\n    background: #ffffff;\n    color: #313030;\n}\n</style>\n\n<table>\n<thead>\n<tr>\n    <th>TYPE</th>\n
<th>ARTIFACT<br></th>\n    <th>REPUTATION</th>\n</tr>\n</thead>\n<tbody><%for item in format_table%> \n<tr>\n   <td><<item.type>>
<br></td>\n   <td><<item.item>><br></td>\t\t\n   <td><<item.reputation>><br>
</td>\n</tr>\n<%endfor%>\n</tbody>\n</table>\n<br>\n\nBest Regards,<br>\nCommSec Tines Automation"
}
```

Figure 7: Analysis Result Formation

5.4 Reporting

The consolidated results from the analysis workflow are formulated using hypertext markup language in a tabular format. The results are then sent over email to the email recipient for further lookup. The analysis results (figure 9) categorize each IOC in the associated category with their respective reputation scores.

6 Evaluation

The evaluation of this research was to address the research question discussed in section 1.2 of this report. The main objective of the evaluation process was to verify if the threat-hunting process could be automated for suspicious emails consisting of obfuscated file attachments while extracting all potential IOCs from the email. Also, another objective was to verify if the mean time to detect metric (MTTD) for analysis can be minimized. To evaluate the presented automated phishing analysis workflow, three (3) experiments were conducted which are discussed in the following sub-sections.

6.1 Case Study 1: Analysing Benign Email

The primary objective of this research was to automate the overall threat-hunting process for suspicious emails and the successful extraction of associated IOCs followed by the determination of their respective reputation scores. However, the first and foremost consideration was to analyze a benign/legit email against the automated workflow and observe its behavior. A legit email (figure 8) attached with a text file and random text data with a *google.com* link in the email body was forwarded to the analysis workflow.

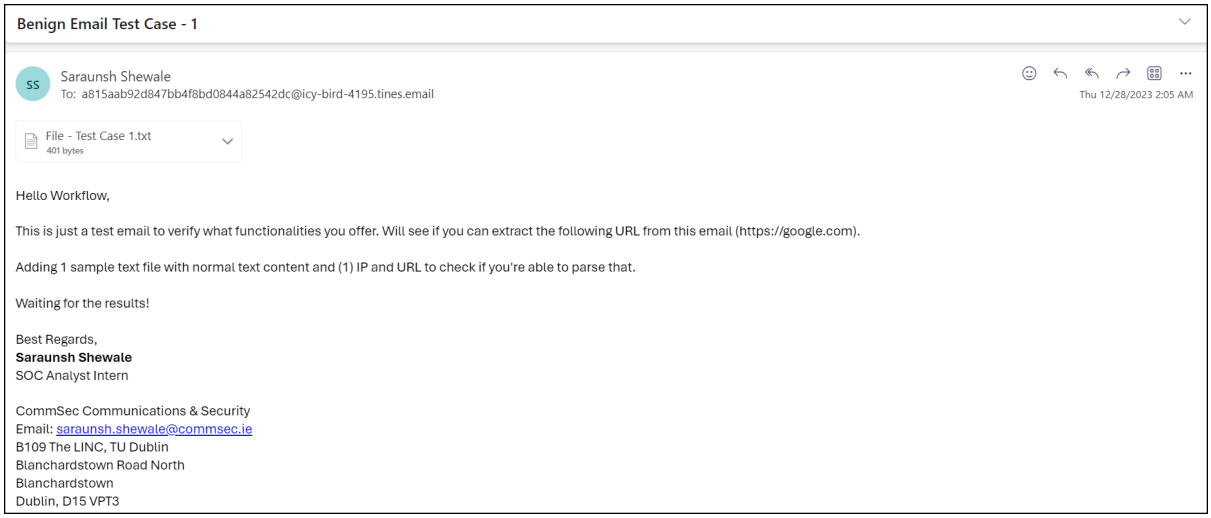

Figure 8: Case Study 1 - Benign Email

The analysis was reported back to the same email address in 100 seconds. The reported results demonstrated the holistic interpretation and extraction of associated IOCs with their credibility scores. The workflow detected the *google.com* URL successfully with a safe reputation score as intended. The attached file was also parsed and analyzed accurately as shown in figure 9.

Suspicious Email Analysis Results - 28 Dec, 2023

Hello Team,

Thanks for sending the email to suspicious mail box. See the attached results from the analysis.

Please note: Zero (0) reputation score means the IOC is clean/safe.

TYPE	ARTIFACT	REPUTATION
Static File Analysis IP	205.251.242.103	0
Attachment	File - Test Case 1.txt	No Record Found
Static File Analysis URL	hxxps://facebook[.]com	1
IP Address	198.154.180.224	0
IP Address	10.167.16.182	0
IP Address	40.107.21.129	0
IP Address	104.47.18.105	0
Email	saraunsh.shewale@commsec.ie	Not Supicious
URL	hxxps://google[.]com	No Record Found

Best Regards,
CommSec Tines Automation

Figure 9: Case Study 1 - Analysis Results

6.2 Case Study 2: Malicious Phishing Email

In this experiment, a phishing email sample was collected from the open-source GitHub[8] repository which consists of hundreds of phishing email samples for analysis. One phishing email (figure 10) from multiple samples was forwarded to analyse its associated artifacts.

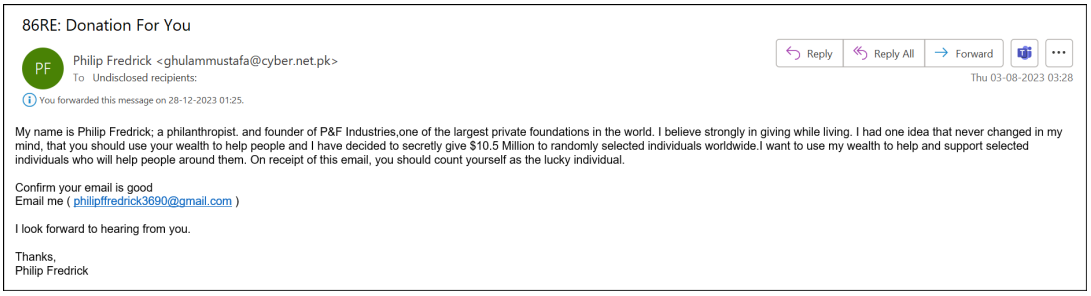

Figure 10: Case Study 2 - Phishing Email

The analysis results (figure 11) were successfully able to extract all potential IOCs from the phishing email. Based on analysis, it flagged one of the email addresses as suspicious due to its involvement in illicit operations.

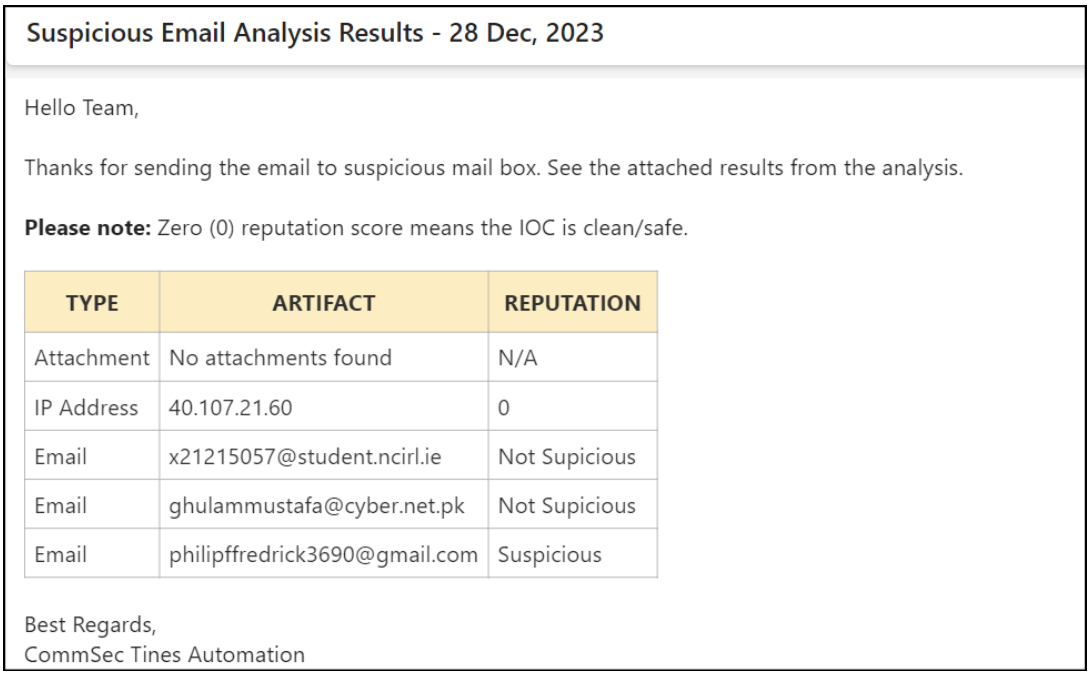

Figure 11: Case Study 2 - Analysis Results

6.3 Case Study 3 - Analysing Obfuscated JS Code

This experiment was conducted to evaluate if the obfuscated JS code is analysed accurately and if IOCs tied with the packed JS code are identifiable. The obfuscated code is depicted in figure 12.

[8] `https://github.com/rf-peixoto/phishing_pot`

```
(function(_0x1cc00a, _0x2b0014) {
    var _0x2158c5 = _0x3389,
        _0x2ad5fc = _0x1cc00a();
    while (!![]) {
        try {
            var _0x257282 = -parseInt(_0x2158c5(0x1b6)) / 0x1 * (-parseInt(_0x2158c5(0x1b5)) / 0x2) + parseInt(_0x2158c5(0x1b1)) / 0x3 + parseInt(_0x2158c5(0x1ae)) / 0x4 * (
            parseInt(_0x2158c5(0x1b3)) / 0x5) + -parseInt(_0x2158c5(0x1b0)) / 0x6 * (parseInt(_0x2158c5(0x1b2)) / 0x7) + parseInt(_0x2158c5(0x1ad)) / 0x8 *
            (-parseInt(_0x2158c5(0x1af)) / 0x9) + -parseInt(_0x2158c5(0x1ba)) / 0xa * (parseInt(_0x2158c5(0x1ac)) / 0xb) + -parseInt(_0x2158c5(0x1b7)) / 0xc;
            if (_0x257282 === _0x2b0014) break;
            else _0x2ad5fc['push'](_0x2ad5fc['shift']());
        } catch (_0x4648b1) {
            _0x2ad5fc['push'](_0x2ad5fc['shift']());
        }
    }
}(_0x17f3, 0x32574));

function obfuscate() {
    var _0x7c6bd8 = _0x3389;
    console[_0x7c6bd8(0x1b9)](_0x7c6bd8(0x1b4)), document[_0x7c6bd8(0x1b8)](_0x7c6bd8(0x1bb));
}

function _0x3389(_0x3b4d80, _0x26bf5b) {
    var _0x17f38e = _0x17f3();
    return _0x3389 = function(_0x338923, _0x2e9a22) {
        _0x338923 = _0x338923 - 0x1ac;
        var _0x557a52 = _0x17f38e[_0x338923];
        return _0x557a52;
    }, _0x3389(_0x3b4d80, _0x26bf5b);
}
obfuscate();

function _0x17f3() {
    var _0x1117dc = ['https://google.com', '198ytXiwT', '16RFicpd', '586504aKFEFA', '1673442WlhcYy', '690Udrnts', '6579060QQKsu', '2891GqMKIC', '10htUGrj', '80.67.172.162',
    '697078SoIsMA', '1jcDReL', '22740002kBrdG', 'write', 'log', '25570OuURUK'];
    _0x17f3 = function() {
        return _0x1117dc;
    };
    return _0x17f3();
}
```

Figure 12: Case Study 3 - Obfuscated Code Snippet

The OpenAi API integration did a great job by simplifying the obfuscated JavaScript code and passing it to regex event transform actions in the storyboard. The IOCs from the code were successfully retrieved and assessed to calculate their reputation score from the VirusTotal threat intel service. The results from the analysis are depicted in the figure 13

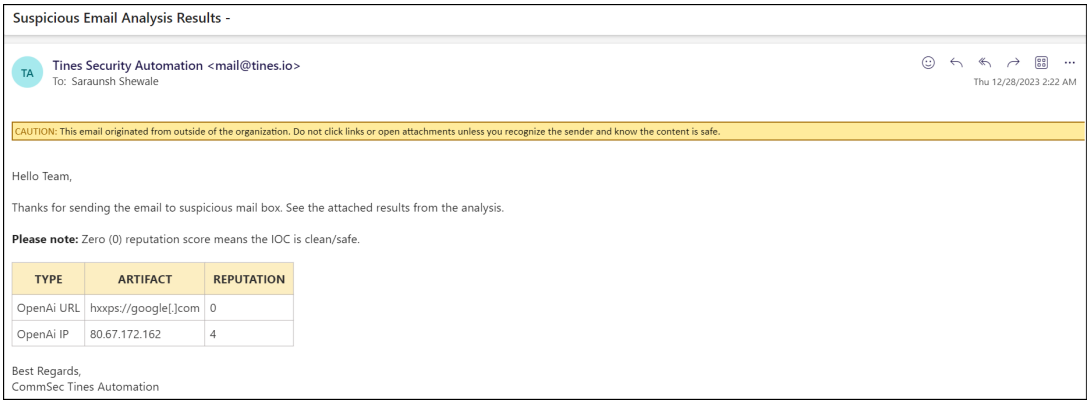

Figure 13: Case Study 3 - Analysis Results

6.4 Discussion

The use of popular threat intel services such as VirusTotal, URLScan, EmailRep and the most powerful being OpenAi supported the automated phishing analysis workflow designed using cloud-based SaaS service, Tines. The automated analysis results were observed to be accurate and efficient in terms of extraction of IOCs from suspicious emails followed by its analysis. Three (3) experiments were performed based on benign, malicious, and obfuscated phishing samples and all of these experiments worked as intended. One limitation of this research was that the occurrence of different character (UTF) encodings in email could interrupt the analysis workflow. This can be improved by adding exception-handling mechanisms to the existing architecture.

One more limitation of this research was found in OpenAi's API integration. For the same obfuscated code, there can be multiple random answers because of model temperature and seed value settings. There are some limitations in integrating the automatic code simplification logic in the primary analysis workflow due to the requirement of manual intervention when submitting the obfuscated code for analysis. As the research is utilizing free API credits for this service and thus it cannot operate cost-effectively on lower temperature settings. There were some limitations observed for the URLVoid API service in terms of its response speed which slowed down the analysis performance. Some minor errors while IOC extraction were observed. Specifically, the unnecessary identification of legit artifacts such as Outlook's mail server IP address, and the original sender's email address. These can be omitted by defining a valid regex filter.

7 Conclusion and Future Work

The research suggests that the use of such automated workflows for suspicious email analysis saves time and a lot of effort. It was able to successfully detect and analyse all suspicious artifacts from email and it is proven to be a reliable approach based on (3) case study outcomes that covered a sufficient number of possibilities. However, a limited set of phishing emails was tested against the automated workflow considering the cloud service bandwidth in terms of API calls. Some of the observed limitations of this research are as follows -

1. Encounter of different UTF character encoding in email body can interrupt the flow of analysis and result in less reliable outcomes.

2. OpenAi could result in random answers for the same question prompt due to temperature and seed value settings for the model.

3. False detection and extraction of legit, safe artifacts from emails such as identification of Outlook's mail server IP and analysis initiator email address.

4. Performance slow-down issues due to the usage of delay event transforms because of URLScan's API performance limitations.

The research did not address the above-mentioned limitations effectively and leaves a scope for more future work. Overall, the use of automated analysis workflow appears to be valuable, but it should be used in conjunction with semi-automatic tools and manual analysis techniques. There is a need for additional advancements and research to address the following challenges -

1. Additional consideration of the latest API services in the threat intelligence area while tweaking the model configurations of existing services to achieve greater efficacy.

2. The lack of support for different UTF character encoding schemes leaves further room for additional support integration.

References

Búadóttir, T., Mascio, O. and Eckroth, J. (2023). Kira: A financial chatbot using chatgpt and data obfuscation, *J. Comput. Sci. Coll.* **39**(3): 277–294.

Choi, Y., Kim, T., Choi, S. and Lee, C. (2009). Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis, *in* Y.-h. Lee, T.-h. Kim, W.-c. Fang and D. Ślezak (eds), *Future Generation Information Technology*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 160–172.

Cova, M., Kruegel, C. and Vigna, G. (2010). Detection and analysis of drive-by-download attacks and malicious javascript code, *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, Association for Computing Machinery, New York, NY, USA, p. 281–290.
URL: *https://doi.org/10.1145/1772690.1772720*

Dujmović, T., Skendrović, B., Kovačević, I. and Groš, S. (2023). Detection and analysis of obfuscated and minified javascript in the croatian web space, *2023 46th MIPRO ICT and Electronics Convention (MIPRO)*, pp. 1252–1257.

Fang, Y., Huang, C., Liu, L. and Xue, M. (2018). Research on malicious javascript detection technology based on lstm, *IEEE Access* **6**: 59118–59125.

Galdi, E., Perrone, G. and Romano, S. P. (2022). Automated open-source phishing email analysis platform, *Proceedings of the Italian Conference on Cybersecurity (ITASEC 2022)*.
URL: *https://ceur-ws.org/Vol-3260/paper6.pdf*

He, X., Xu, L. and Cha, C. (2018). Malicious javascript code detection based on hybrid analysis, *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 365–374.

Hou, B., Yu, J., Liu, B. and Cai, Z. (2018). Jspre: A large-scale detection of malicious javascript code based on pre-filter, *in* X. Sun, Z. Pan and E. Bertino (eds), *Cloud Computing and Security*, Springer International Publishing, Cham, pp. 586–599.

Kılıç, E. and Sandıkkaya, M. T. (2023). Obfuscated javascript code detection using machine learning with ast-based syntactic and lexical analysis, *2023 8th International Conference on Smart and Sustainable Technologies (SpliTech)*, pp. 1–6.

Likarish, P., Jung, E. and Jo, I. (2009). Obfuscated malicious javascript detection using classification techniques, *2009 4th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 47–54.

Lu, G. and Debray, S. (2012). Automatic simplification of obfuscated javascript code: A semantics-based approach, *2012 IEEE Sixth International Conference on Software Security and Reliability*, pp. 31–40.

Masri, R. and Aldwairi, M. (2017). Automated malicious advertisement detection using virustotal, urlvoid, and trendmicro, *2017 8th International Conference on Information and Communication Systems (ICICS)*, pp. 336–341.

Morishige, S., Haruta, S., Asahina, H. and Sasase, I. (2017). Obfuscated malicious javascript detection scheme using the feature based on divided url, *2017 23rd Asia-Pacific Conference on Communications (APCC)*, pp. 1–6.

Pressman, R. S. (2010). *Process Models*, seventh edn, McGraw-Hill Higher Education, p. 41–42.

Saunders, M., Lewis, P., Thornhill, A. and Bristow, A. (2019). *"Research Methods for Business Students" Chapter 4: Understanding research philosophy and approaches to theory development*, pp. 128–171.

Srndic, N. and Laskov, P. (2013). Detection of malicious pdf files based on hierarchical document structure, *Network and Distributed System Security Symposium*.
URL: *https://api.semanticscholar.org/CorpusID:1458246*

Starov, O., Zhou, Y. and Wang, J. (2019). Detecting malicious campaigns in obfuscated javascript with scalable behavioral analysis, *2019 IEEE Security and Privacy Workshops (SPW)*, pp. 218–223.

Talukder, M., Islam, S. and Falcarin, P. (2019). Analysis of obfuscated code with program slicing, *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pp. 1–7.

Wang, J., Xue, Y., Liu, Y. and Tan, T. H. (2015). Jsdc: A hybrid approach for javascript malware detection and classification, *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, Association for Computing Machinery, New York, NY, USA, p. 109–120.
URL: *https://doi.org/10.1145/2714576.2714620*

Xu, W., Zhang, F. and Zhu, S. (2012). The power of obfuscation techniques in malicious javascript code: A measurement study, *2012 7th International Conference on Malicious and Unwanted Software*, pp. 9–16.