Web Defacement Monitoring Tool
Project Description Document

J.Hou
3565155
Robert Sobukwe Rd, Bellville
Cape Town, 7535
3565155@myuwc.ac.za

ABSTRACT

This report focuses on development of a tool to minimize damage due to website defacements. The report follows the format of historical defacement events and other studies in the same field. The user requirements analysis states the web administrator perspective and the requirements analysis are broken down into system comments and software elements.

This project is sponsored by CSIR to educate users about web defacements and cyber security awareness in general through developing a tool to prevent such hacktivism from happening.

KEYWORDS

Web defacement, cyber security, data breach, denial of service, Web Defacement Monitoring Tool, WDMT.

1 INTRODUCTION

The advancement of the Web and the applications inspired new methods to communicate and share information, nearly all businesses and organizations utilize web pages to communicate with end-users. The Internet contains vast amount of web pages and information, a web page is a Hypertext Markup Language (HTML) document, typically a web site has many web pages linked, hosted on a web server [1].

A website contains information and critical data related to the organization, this typically gains audience from end-users and unwanted attention from hackers. The earliest examples of hacktivism and web defacement date back to 1996, the United State Department of Justice web server was hacked and defaced, hackers replaced the US Department of Justice website homepage with a text “Department of Injustice” and display of pornographic content [2]. More recent examples are, 2018 July 7 a report by News24 the Presidency government website was hacked and defaced by a hacker Black Team, the website was change with the text “Hacked By Black Team. Sahara is Moroccan. And Morocco is ur Lord!” [3] [4].

Web security are crucial to protect organizations in this digital era, hackers are improving and exploiting any possible loop hole, its therefore essential to practice and use web defacement monitoring tools.

2 LITERATURE REVIEW

Ebot Ebot Enaw and Djoursoubo Pagou Prosper proposed a Conceptual Approach to Detect Web Defacement through Artificial Intelligence [5]. The authors discuss the use of artificial intelligence concepts such as anomaly detection, machine learning and inferences to detect web defacement and unauthorized access [5]. The authors provided an intelligent way to efficiently detect the signature (type) of a web defacement attack, the detection algorithm is consistently self-improved [5]. The authors designed a new architecture to learn criteria that is used to characterize websites, through this the tool studies the behaviors of a normal web site [5]. A web defacement trainer module was developed to analyze previous web defacement signatures, and based on these signatures improves the accuracy of a web defacement attack [5]. The web crawler and analyzer module work in cohesion, the result is then compared to the result of a normal behavior module through given criteria [5]. The detection algorithm sends an email notification to the web administrator with the log if a web defacement is confirmed [5].

Tushar Kanti proposed Implementing a Web Browser with Web Defacement Dection Techniques [6]. The author discussed through the use of detection techniques and developed a web browser to enhance the detection accuracy of web defacement attacks [6]. The checksum is calculated through comparison of the current webpage with the backed-up website and if there’s a difference in checksum, this means defacement occurred. The difference algorithm is called when defacement is detected, the functionality of the difference algorithm is to spot the exact location of the defacement, comparing with the original backed up version [6]. The web browser was modified based on Internet Explorer, the user will see no difference while browsing, the web browser will notify the web administrator of any detection of web defacements [6]. The author concludes the report with a recovery mechanism for the defaced web pages, the use of a difference algorithm to spot the exact location of defacement and the use of a checksum to replace the defaced html code [6].

3 PROJECT PROPOSAL

Web defacement is defined as unauthorized changes to a website such as text, company logos, images, and videos. In some cases, the entire website is completely changed. The perpetration is usually wanting to distribute a message (advertisement, malware link, etc.) or they may simply make fun of the website owner. In some cases, the defaced website may display offensive messages or information to viewers and customers. Web defacement is considered a trivial crime however it has the following implication on an organization and sometimes with lasting effects: humiliation and damage to reputation, services disruption and information downtime and potential data breach.

The objective of the project is to prevent web defacement through constructing a Web Defacement Monitor Tool (WDMT). The web administrator should install the corresponding tools, the WDMT will scan web file contents and its media, compare to the backed-up files by the use of hashing and checksum calculation. Detected defacement will be logged and send to the web administrator via email in report form, the WDMT will restore the damaged content from the backup, a further report will be sent after the restoration, logging defacement details.

4 USER REQUIREMENTS

4.1 About the Project

The internet consists of vast amount of information, a common way to access this information is through the website. These days many users make use of websites, popular websites attract unwanted attentions for hacktivism.

This project was proposed by Council for Scientific and Industrial Research (CSIR) as a collaborative work with the University of Western Cape (UWC) honours students. The aim of this project is to educate the user about what is web defacement, develop a web defacement monitoring tool and recover defaced sites by backups, the tool developed should be automated, identified defacement should be notified via log or screenshot in a document report form.

4.2 User view of the Project

The web administrator regulates and maintains web systems. In this project the web administrator must protect the website against defacement, defacement have negative impact on an organization, lost in public reputation, system down time and potential data breaches.

It is essential to have counter measurements against defacement; the objective of this project is to create a defacement tool that automatically detects web defacement and repairs damage content in a reasonable time.

4.3 Problem Domain

The problem domain in this project is understanding the HTML and web hosting, the website might be host online through a service provider, the user should be familiar with the terms and operations needed to implement the web defacement monitoring tool.

4.4 Expectations

The WDMT is expected to minimize damage due to web defacement on an organization this is accomplished through making a backed-up version of the web site. The WDMT is an automated tool that checks the current web site with the backed-up web site, ensuring security and quality of the web site. The WDMT should automatically repair damaged content once web defacement was detected, appropriate reports will be sent to the web administrator during detection and after repair phase of the tool.

4.5 Stakeholder Requirements

In this project, the stakeholder CSIR defined the following requirements, the user group consists of web administrators, technician that create and maintain websites, the WDMT must include the following functionalities:

- Create backups of a website files contents, images, videos and web text
- Instantly detect when a website has been defaced and notify the web administrator
- Restore damage content in the shortest possible time.
- The mentioned functions of web defacement detecting tool should all be automated.

Other user groups are end users that test the WDMT or visit the web site for various information's, these users have no access to files for security measurements.

4.6 Limitations and Out of Scope

The WDMT doesn't perform the following functionalities:

- Track access information of end-user, such as IP address, MAC address and other system information.
- Reverse tracking and attempt to access end-user privacy.

4.7 User requirements for Web Administrator

The Web Administrator must do the following to ensure the software is operated correctly:

- Web Admin must create website with media content such as video and images.
- Install and implement WDMT.
- Using the WDMT, backup website and content with hashing, comparing checksum method.
- Set WDMT defacement automated scan interval example: 10 minutes.
- Set profile information such as name and email.

4.8 User Case Diagram

Figure 1 below shows the relationship between the Web Administrator and the end-user; the Web administrator will regulate the website and quality of the service while the end-user accesses the website for information.

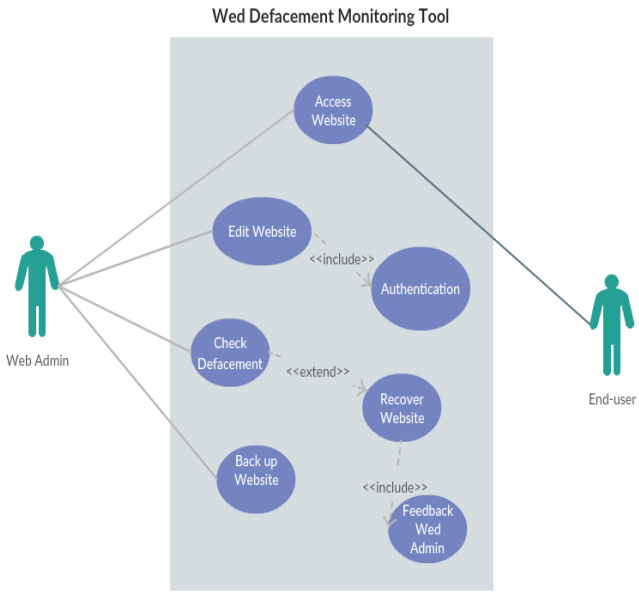

Figure 1 Use Case Diagram of WDMT

5 REQUIREMENTS ANALYSIS

5.1 Current System

The WDMT is being developed on a Linux platform, prerequisite software's are Chrome or any web browser software, Python 3.5.2, GitHub and Git 2.7.4.

Figure 2 below shows the Linux environment on the right and temporary website used to test against web defacement on the left.

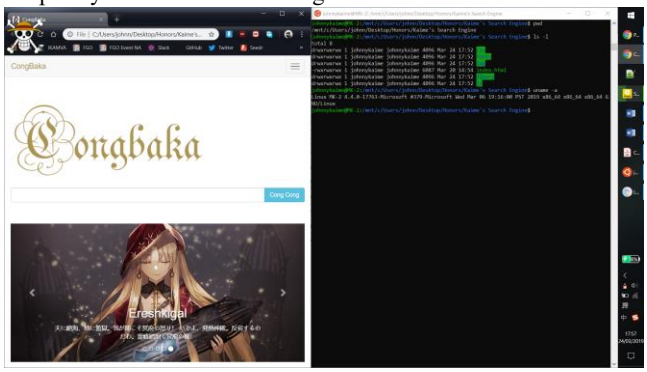

Figure 2 Current System used to develop WDMT

5.3 Functional Requirements

- The WMDT must back-up the current undefaced web site.
- The WDMT must detect web defacement attacks. Attacks include any modification or missing of text, graphic images or video.
- The WDMT must provide feedback in report form and email to the web administrators. First report when web defacement was detected.
- The WDMT must recover damage site through comparing hashed checksums with the backed-up website.
- The WDMT must provide a secondary feedback report after restoration of damage web sites.

5.4 UML Class Diagram

Figure 3 below shows the UML class relationship, the WDMT will be developed under an Object-orientated programming (OOP) approach.

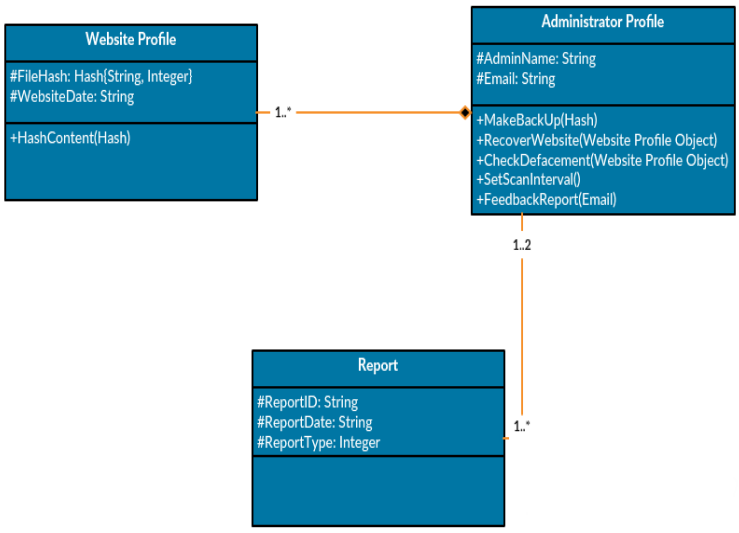

Figure 3 UML Class Diagram

6 NON-FUNCTIONAL REQUIREMENTS

6.1 Performance Requirements

The WDMT scan should be automated and each scan in attempt to identify defacement should not take longer than 5 minutes.

6.2 Operating Requirements

Compulsory software is Python 3 and Linux platform is required to operate the WDMT.

6.3 Reliability

Reliability to WDMT is a big concern, the WDMT should be always check input and filename to prevent human error, WDMT

will prompt feedback if files of corresponding filenames are not found.

6.4 Usability

The WDMT will be a terminal line base software, the web administrator can interact the software using number such as 1, 2, 3 etc. Each number corresponds to a different functionality, a menu mapping will be provided in the command menu interface for more ease of access.

7 USER INTERFACE

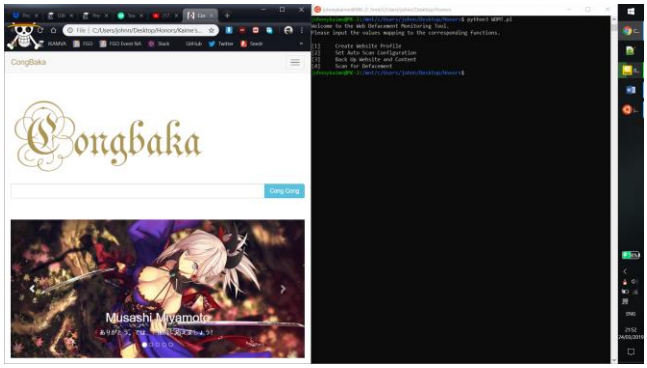

Figure 4 Interface of Website and WDMT

The web administrator is expected to create a website containing media such as videos and images. The above figure 4 shows a website used as a test against the WDMT.

The main interface for the WDMT is command line based, it operates through a terminal in Linux or Bash terminal on Windows 10. The web administrator will navigate the WDMT via numbers, in the above figure 4, the web administrator would have to enter 4 in the terminal to force scan any web defacements, provided all the necessary steps are completed.

8 DESIGN

8.1 Data/Function Diagram

The WDMT main functionality can be decomposed into sub functions, the below figure 5 represents the relationship between each function.

The WDMT will require the Administrator input for the first functionality, the Administrator Creates Website Profile, core information such as hash string of each file, location of each file and Auto Scan Configuration are stored.

The second function Backup Website creates a copy of the website and all content, this is stored in a different location for further use.

The third function Monitor Websites happens automatically between each period interval, the purpose is to hash a file and find a mismatch in hash string when comparing to the Website Profile created in the function Create Website Profile. Once a defaced file is detected a byte-to-byte comparison is done to find the exact defaced location, the detail information is sent to the fifth function to generate a Defacement Report to the Administrator. After all defaced files are detected the WDMT recovery damage content through the Backup Website and a further information log is passed onto the fifth function, the Recovery Report is then sent to the Administrator.

The fourth function Scan for Defacement, calls the third function to do an immediate hash comparison and proceed from that point onwards.

The fifth function Generate Reports, this function is called by other functions as a feedback mechanism, only if a defacement is detected the Defacement Report will be sent, after the recovery of damage content the Recovery Report will be sent.

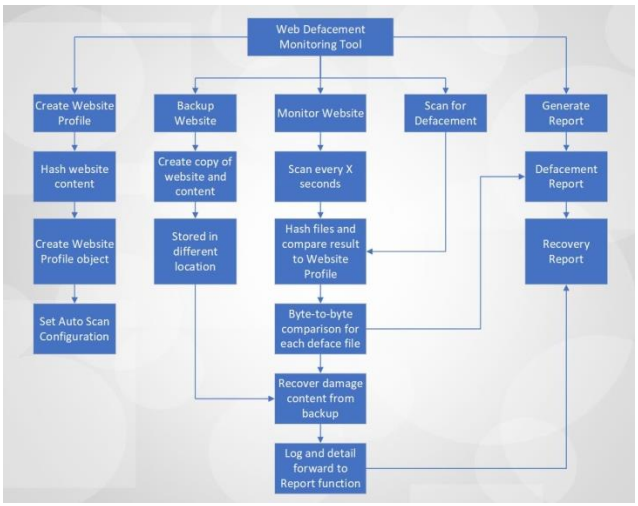

Figure 5 Functional Decomposition of the WDMT

8.2 Architectural Design

The architecture looks at the High-Level Design (HLD) of the WDMT. The below figure 6 shows the high-level design of the WDMT and the process follow.

The first section involves the administrator to either use the terminal and command line or the GUI to create a website profile of the target website, followed by configuring the auto scan configurations.

The second section involves the WDMT and the target website, the WDMT would scan the website files and compare its hash value with the result stored in the website profile.

The third section triggers if the second section is true and the WDMT compares line by line with the text or document file types, other techniques will be used to compare image and media format files.

The last section involves feedback, this is achieved by the WDMT creating a report then send to the Administrator, detailed description to which content was defaced and when the defacement took place.

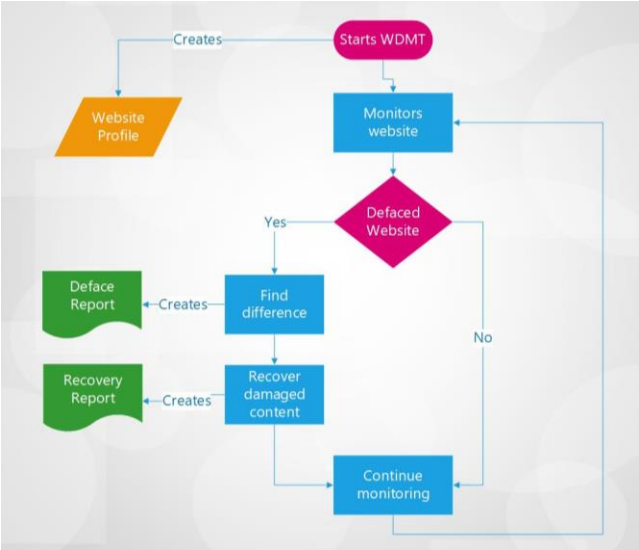

Figure 6 High-Level Design

8.3 Interface Design

8.3.1 Terminal and command line. The Administrator will interact the WDMT through the terminal and use numbers corresponding to each menu option to trigger the different functionalities.

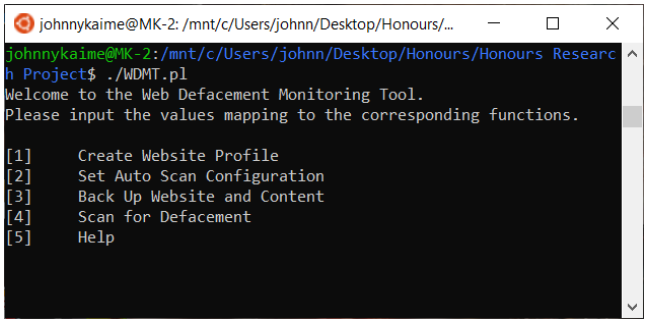

Figure 7 Terminal and command line interface

The above figure 7 shows the terminal and command line interface. The Administrator will input for example 1 to trigger the event function corresponding to that input, other input such as strings and mismatch will be caught, the Administrator will then require to input valid inputs.

8.3.2 Graphical User Interface (GUI). The Administrator may alternatively use the graphical interface for a better experience, the Administrator will click on a button to trigger the corresponding event function.

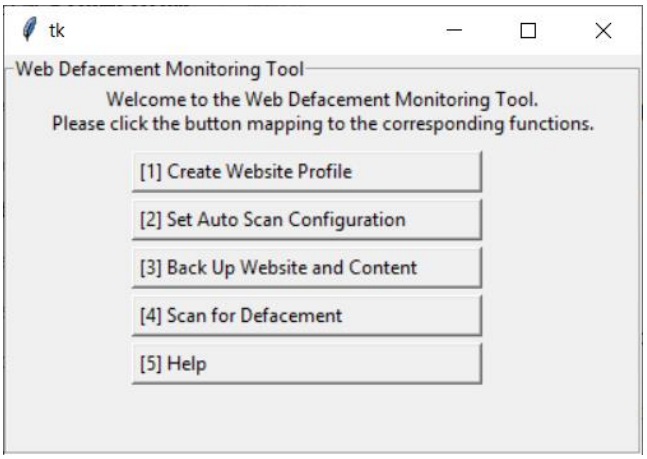

Figure 8 Graphical User Interface

The above figure 8 shows the GUI of the WDMT. The Administrator will click on the button to trigger the event function, the GUI validates the input to a certain level, because the Administrator is restricted to clicking.

8.4 Interface Functionality

8.4.1 Create Website Profile. The Administrator will select the root directory of the website, the WDMT will scan all the files and make a profile of total files, name of the html pages and media file formats etc.

8.4.2 Set Auto Scan Configuration. The Administrator will use this to configure the WDMT auto scan function, variables such as waiting time between scans etc.

8.4.3 Backup Website and Content. The WDMT creates a backup of the website from the Website Profile. Backup Website and Content will only work if a Website Profile has been created.

8.4.4 Scan for Defacement. The WDMT will immediately scan the website and compare the hash result with the website profile to find any defacement, if any defacement was detected, the WDMT will auto recover the deface content.

*8.4.5 Help. T*he Help function explains how to use each function and credits by the author.

8.5 Component Level Design

The component level design looks at the technical aspect to the WDMT, below the figure 9 shows the Low-Level Design of the WDMT.

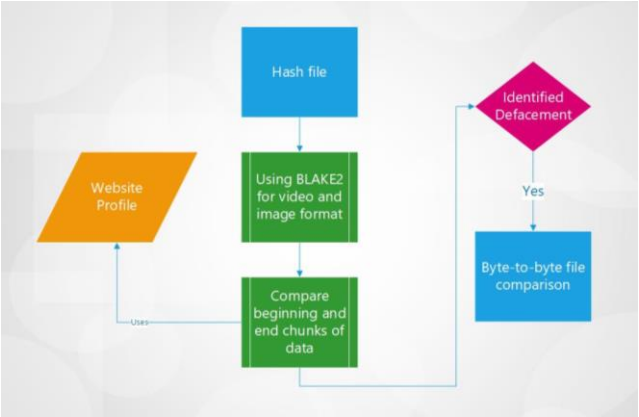

Figure 9 Low-Level Design

The main focus in low-level design is the function Hash file, hashing a file provides a unique hashed signature used to compare if the file has been modified, the core hashing algorithm of the WDMT will be BLAKE2, once defacement has been detected a byte-to-byte comparison is performed to find the exact location of the defaced content.

The below figure 10 shows the SHA1 hashing algorithm,

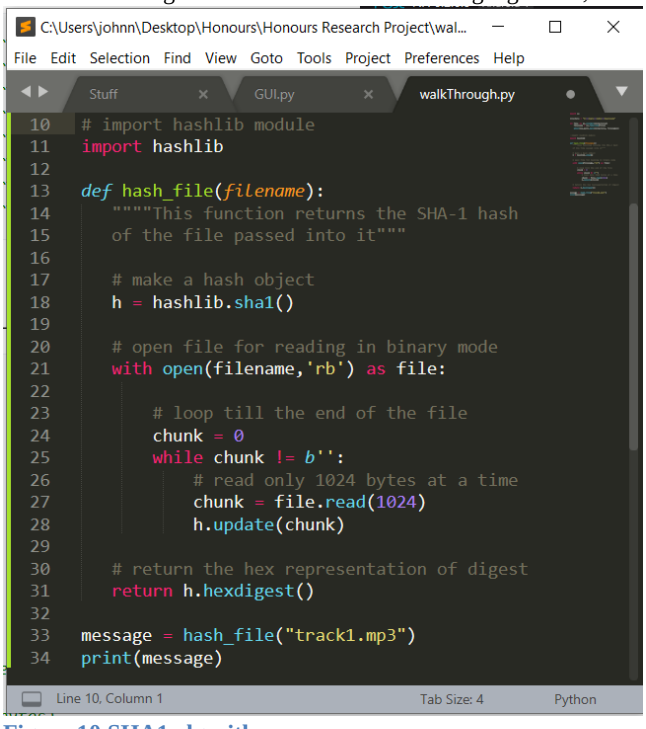

Figure 10 SHA1 algorithm

The SHA1 hashing algorithm was used during research to compare the different speed and performance, the above example would be applied to any object file, it is essentially important to note that all incoming files are converted into byte objects before updating the data chunk.

The hashing algorithm. BLAKE2 will be our main hashing algorithm for WDMT, research show BLAKE2b, a 64-bit hashing algorithm is slightly faster than and secure as SHA3.

Each time we compare a file or any data BLAKE2 requires each byte object conversion of the target, each byte object is then updated to our BLAKE2 constructor with the preset chunk size, digest size and parameters. Once the BLAKE2 hash all the bytes object in a file it outputs a 128-character hash signature that's unique to the target file. From the figure 11 below shows a usage of BLAKE2 algorithm, hashing the simple String "Hello world", notice the String object is converted into a byte object before passing into the BLAKE2 update function.

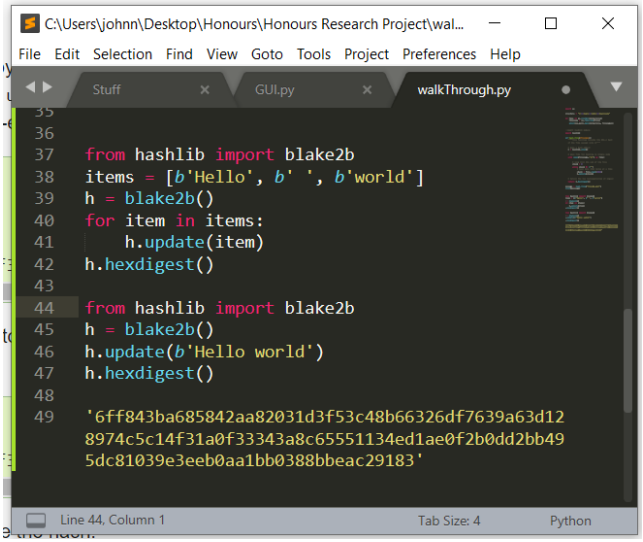

Figure 11 BLAKE2 usage

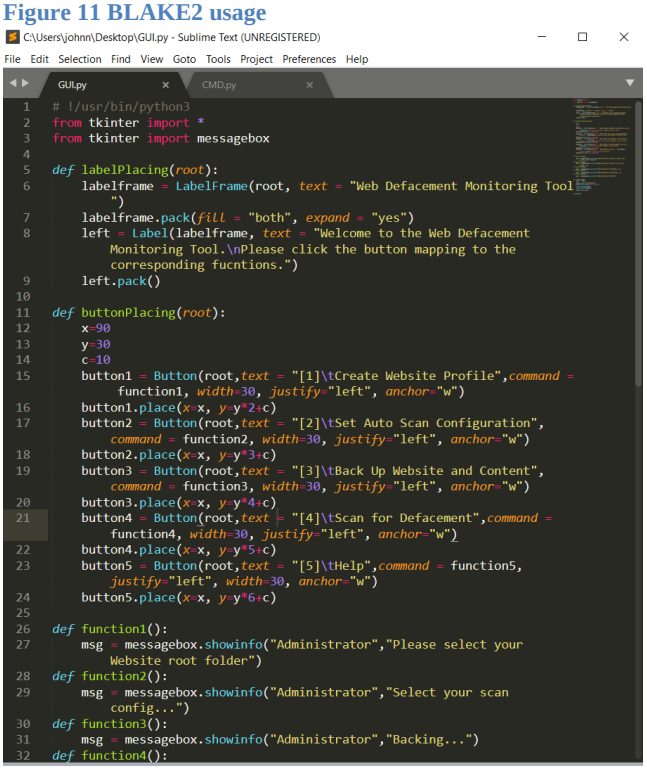

Figure 12 Python code of Prototype

The above figure 12 shows the Python code of the WDMT prototype, it utilizes both SHA and BLAKE2 as its core hashing algorithm.

The BLAKE2b algorithm is an adapted algorithm based on BLAKE, it performs optimal on a 64-bit machine, each digest size is 64-bit, the algorithm utilizes modern processing architecture.

The WDMT will take an object orientated approach the Administrator creates a Website Profile that includes Website root directory names, total amount of files, a Hash Object (created using BLAKE2 constructor) and a long string with all file's names and directories, refer to Appendix B for more information.

9 PROTOTYPE

The WDMT is implemented by using the following elements.

9.1 Hardware

The hardware elements are a computer or server is the only requirement for the WDMT, it is ideal to have a powerful server when hosting the website and using the WDMT, as the performance and speed of hashing is high affect by the process and memory of the machine.

9.2 Software

The WDMT will be developed on a Linux based system, currently using the Linux Subsystem for Windows 10 for development.

Python 3.6 and above will be used, packages from Python3 such as pyblake2 library use for BLAKE2 algorithm hashing and constructing hash object, hashlib library for other common hashing algorithms and object, os library for directory navigation and file extraction.

The main program of WDMT will be coded in Python and the prototype will be tested against our own University of the Western Cape Computer Science server, CSUNX.

Apache2 will be used for web hosting on the local machine during development phase and also on the CSUNX server during testing phase.

Hyperlink references

REFERENCES

[1] M. Masango, F. Mouton, P. Antony and B. Mangoale, "Web Defacement and Intrusion Monitoring Tool: WDIMT," September 2017.

[2] D. Dorothy, "Georgetown Journal of International Affairs," *The Rise of Hacktivism,* 2015 September 2015.

[3] M. Mxolisi, "Presidency website up and running after hacking attack," News24, 7 July 2018. [Online]. Available: https://www.news24.com/SouthAfrica/News/breaking-presidency-website-hacked-20180707. [Accessed 6 March 2019].

[4] N. Mphathi, "SA Presidency website hacked," Independent Online (IOL), 11 July 2018. [Online]. Available: https://www.iol.co.za/dailynews/news/sa-presidency-website-hacked-15950026. [Accessed 6 March 2019].

[5] E. E. Enaw and D. Pagou Prosper, "A Conceptual Approach to Detect Webdefacement Through Artificial Intelligence," *International Journal of Advanced Computer Technology (IJACT),* vol. 3, no. 6, pp. 77-83.

[6] K. Tushar, "Implementing a Web Browser with Web Defacement," *World of Computer Science and Information Technology Journal (WCSIT),* vol. 1, no. 7, pp. 307-310, 2011.

APPENDIX

A Project Plan Gantt Chart

B BLACKE2b Algorithm

```
Algorithm BLAKE2b
   Input:
      M            Message to be hashed
      cbMessageLen: Number, (0..2^128)
Length of the message in bytes
      Key
Optional 0..64 byte key
      cbKeyLen: Number, (0..64)
Length of optional key in bytes
      cbHashLen: Number, (1..64)
Desired hash length in bytes
   Output:
      Hash
Hash of cbHashLen bytes

   Initialize State vector h with IV
   h0..7 ← IV0..7

   Mix key size (cbKeyLen) and desired
hash length (cbHashLen) into h0
   h0 ← h0 xor 0x0101kknn
         where kk is Key Length (in
bytes)
                nn is Desired Hash
Length (in bytes)

   Each time we Compress we record how
many bytes have been compressed
   cBytesCompressed ← 0
   cBytesRemaining  ← cbMessageLen

   If there was a key supplied (i.e.
cbKeyLen > 0)
   then pad with trailing zeros to
make it 128-bytes (i.e. 16 words)
   and prepend it to the message M
   if (cbKeyLen > 0) then
      M ← Pad(Key, 128) || M
      cBytesRemaining ←
cBytesRemaining + 128
   end if

   Compress whole 128-byte chunks of
the message, except the last chunk
   while (cBytesRemaining > 128) do
      chunk ← get next 128 bytes of
message M
      cBytesCompressed ←
cBytesCompressed + 128  increase count
of bytes that have been compressed
      cBytesRemaining  ←
cBytesRemaining  - 128  decrease count
of bytes in M remaining to be
processed

      h ← Compress(h, chunk,
cBytesCompressed, false)  false ⇒
this is not the last chunk
   end while

   Compress the final bytes from M
   chunk ← get next 128 bytes of
message M  We will get cBytesRemaining
bytes (i.e. 0..128 bytes)
   cBytesCompressed ←
cBytesCompressed+cBytesRemaining  The
actual number of bytes leftover in M
   chunk ← Pad(chunk, 128)  If M was
empty, then we will still compress a
final chunk of zeros

   h ← Compress(h, chunk,
cBytesCompressed, true)  true ⇒ this
is the last chunk

   Result ← first cbHashLen bytes of
little endian state vector h
End Algorithm BLAKE2b
```