

Enhancing Malware Defense in Windows OS Through Ethical Malware Development and Agile-Based Simulation Framework

Nazirah Abd Hamid¹, Siti Dhalila Mohd Satar², Ahmad Faisal Amri Abidin @ Bharun³, Mohd Fadzil Abdul Kadir⁴, Mohammad Afendee Mohamed⁵, Shamil Hakimi Shamsul Bahrin⁶

^{1,2,3,4,5,6}Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Terengganu, Malaysia

* nazirah@unisza.edu.my

Abstract

The widespread use of Windows operating systems has made them a primary target for malware attacks. While numerous studies focus on detecting and defending against such threats, there is limited research exploring the ethical development of malware to better understand its behavior and improve system defenses. This paper addresses that gap by proposing a practical and controlled approach to malware creation, designed specifically for research and educational purposes. The aim is to build and evaluate a non-malicious malware framework that can simulate real-world attack techniques without causing actual harm. Using the Agile development model, the malware was constructed in stages—comprising a dropper, a payload, and evasion methods—and tested in a secure virtual environment. Written in Python and C, the prototype was assessed through antivirus scans and performance stress tests. Results indicated that the malware avoided detection and replicated key attack behaviors, offering valuable insights into existing system vulnerabilities. This work provides a responsible methodology for studying malware in depth and contributes to the advancement of more effective and adaptive cybersecurity strategies.

Keywords: Ethical Malware Development, Windows OS Security, Agile Methodology

1. INTRODUCTION

Windows is one of the most widely used operating systems globally, favored for its user-friendly interface and broad compatibility [1]. However, its popularity also makes it a common target for cyberattacks. As more users rely on Windows OS for daily computing, attackers have developed sophisticated techniques to exploit system vulnerabilities for personal or financial gain. In response, system developers have continuously updated Windows with improved security measures to protect user data and prevent unauthorized access.

In the field of computer science, ethical practices play a critical role in ensuring responsible and constructive use of technology. Adhering to ethical standards helps promote trust, transparency, and accountability—particularly in areas involving user privacy, data handling, and software transparency [2]. One approach to studying system vulnerabilities while maintaining ethical integrity is through the controlled development of non-malicious malware. This method allows researchers to simulate real-world threats in a safe environment, providing valuable insights into how malware behaves and how defense mechanisms can be improved [3].

Malware, short for malicious software, is designed to perform unauthorized actions on a system, often resulting in data theft, financial loss, or system disruption. Common types include viruses, worms, Trojans, ransomware, and spyware [4]. While the creation and distribution of malware are illegal and unethical, studying its behavior through ethical development allows security researchers to stay ahead of evolving threats.

This project aims to explore the Windows OS environment by ethically developing and testing a controlled, non-malicious malware prototype. The development process follows the Agile methodology, enabling continuous improvement through iterative testing and adaptation. The malware prototype will consist of three key components: a dropper, a payload, and evasion techniques, all implemented using Python and C. Testing will take place in a secure and isolated virtual environment to prevent any real-world impact.

By adopting this ethical and methodical approach, the study seeks to contribute to the improvement of Windows OS security and to support the development of more resilient malware detection systems.

The structure of this paper is as follows: Section 2 presents related works on malware and defense mechanisms. Section 3 explains the methodology used to develop the ethical malware framework. Section 4 discusses the results and evaluation, and Section 5 concludes the study.

2. Related works

Due to the illegal nature of malware development, most studies focus on defensive mechanisms rather than the creation of malware itself. However, understanding the principles of malware behavior is essential for strengthening cybersecurity defenses. This section reviews key studies relevant to malware techniques, detection methods, and Windows OS vulnerabilities.

The authors [5] introduced a stealthy malware activation method using binary instrumentation in their study. While innovative, the approach focuses more on avoiding traditional antivirus detection than offering practical mitigation strategies. This paper highlights the growing sophistication in malware techniques, but it lacks broader contextualization within modern threat landscapes. Meanwhile, the research in [6] critically evaluates how malware mutation techniques bypass detection. While the results clearly demonstrate vulnerabilities in current antivirus software, the study is limited by its lack of discussion on countermeasures or defensive improvements. Nevertheless, the findings are highly relevant for research like the present study, which aims to simulate undetected malware.

The authors in [7] provide a technical walkthrough of ransomware encryption using Python. Although it offers practical insight into malware construction, the paper could be strengthened by expanding the ethical implications and real-world applicability of its findings. It is more of a technical manual than an academic analysis. Furthermore, the research in [8] investigated machine learning algorithms for detecting malware in executable files. Their study is rigorous in its methodology and concludes that SVM achieves the highest accuracy. However, the paper primarily deals with detection effectiveness rather than the nature of malware behavior, making it complementary but not directly aligned with ethical malware development.

The study of [9] and [10] both examine OS-level vulnerabilities. While the former takes a broad survey approach, the latter provides empirical testing on different versions of Windows 10. These studies offer foundational knowledge on system weaknesses but would benefit from a deeper exploration of exploit mechanics to better inform malware defense development. Additionally, the researchers [11] proposed the use of dark web reconnaissance to identify emerging cyber threats. Although insightful, the paper's emphasis on data collection rather than actionable countermeasures limits its immediate applicability in malware design or prevention.

The study in [12] explored the controversial application of AI—specifically ChatGPT—in malware creation. While the paper effectively reveals the potential misuse of AI, it does not sufficiently address the safeguards needed to prevent abuse. This underscores a significant gap in AI-related cybersecurity policy and ethics. Finally, the authors introduced an advanced malware detection method using behavior-based analysis. His approach is notable for addressing the limitations of signature-based detection. The study aligns well with the objectives of this project, offering insight into potential defense strategies against the very type of ethical malware developed here [13].

Overall, while each referenced study offers unique insights into malware development and defense, most lean toward detection and system analysis. There is a clear gap in ethically driven malware simulation studies—precisely the area this project aims to address.

3. METHODOLOGY

For this project, the Agile method as in the Figure 1 has been chosen to be implemented in the malware development for a couple of reasons. The first reason is the flexibility and the adaptability of the method. Since the malware development process commences from the ground up, unexpected challenges may occur during its implementation. The functionality of the dropper or the payload may change based on the current knowledge possessed and the security measures that must be bypassed by the malware. Aside from that, the Agile method promotes transparency as the product will always be recorded, tested and reviewed [14]. This will help in keeping the ethical practices of the cybersecurity in check as this development is proposed to test and improve the security countermeasures in Windows OS, not to exploit the weakness for the developer's personal gain. The project will focus on the dropper of the malware first, then the payload and lastly the implementation of evasion technique to bypass the defense mechanisms.

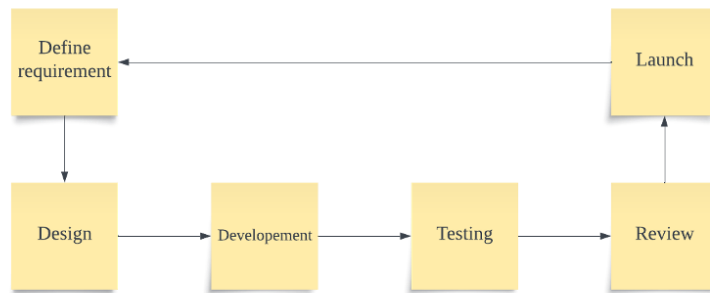


Figure 1. Iterative and incremental development model

The Agile process begins with defining the requirement of the product. Requirements of the product need to be confirmed so the development objective is clear. After that is the design of the product. The design of the product will be proposed with reasons to support it while aligned with the objective. Next are the development and the testing of the product. The product needs to be tested so that the functionality of it is working and to find some errors or bugs. Later, the item then will be reviewed to see if it achieves the objective or needs some upgrade. Lastly, the product will be deployed and monitored [15].

3.1 Framework

There are five stages of the malware framework as shown in the Figure 2. The framework highlights the phases the malware goes through to gain access in the victim's device. Initially, a phishing email will be sent to the victim with a link attachment in it. The link will automatically download an EXE file into the victim's device if clicked on. The EXE file then will be scanned by antivirus to make sure it is legitimate software. The dropper of the malware will be implemented with evasion techniques to avoid detection, analysis and prevention from the security mechanism. The malware will appear legitimate which will trick the victim into executing it. When the malware is executed, the payload will run its function on the victim's device and gain access.

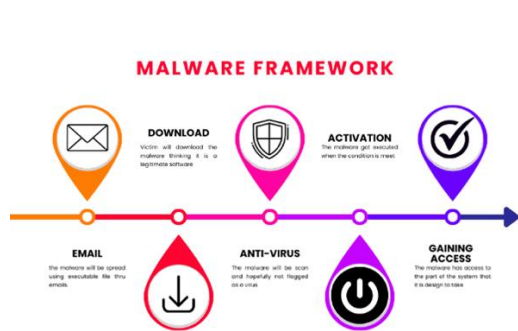


Figure 2. Framework of the proposed study

3.2 Dropper algorithms

The function of the dropper is to deliver the payload to the victim's device. It will appear as a legitimate software to trick the user into executing it. The dropper also plays an important role to bypass the firewall, antivirus (AV) or any other detection system. When the dropper arrives at the victim's device, there are processes that need to be executed for the payload to run its function. In this dropper, the payload file is embedded inside it and encrypted. The key for decryption is also in the dropper in the form of byte. The dropper will read the embedded file which is the payload by getting the .EXE file as stream. After retrieving that, it will read the encrypted file as byte and decrypt it to be able to execute. Once decrypted, the dropper stores the file in a temporary file and sets its permissions to executable. The dropper then will execute the payload at the end of the coding.

The reason why the payload is encrypted in the first place is to be able to evade signature-based detection where the AV scans and reads what the program does based solely on the code. When the payload is

encrypted, the AV can't read the payload code and will not flag it as a malicious file. The payload is encrypted using AES-128 encryption. The keys are randomly generated and cannot be seen when scanned because it is set to be a private variable.

In the dropper algorithm, there are only a few libraries that are important to the code as illustrated in Figure 3. One of them being the cipher and secret key library. These libraries are important to decrypt the encrypted payload using the secret key generated by the same library. Another important library is the 'invocation target exception'. This library allows the code to use Java's reflection or in other words obfuscate function calls within the code. When an AV runs a signature-based scan on the code, it will not be able to know what methods are being called in the code because those methods are not being called directly. This makes the dropper be able to evade this type of security measures easily.

```
import javax.crypto.Cipher;  
import javax.crypto.SecretKey;  
import javax.crypto.spec.SecretKeySpec;  
import java.io.*;  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;
```

Figure 3. Library imported in JAVA for dropper algorithm

There are two main variables in this algorithm as shown in Figure 4. Both are set to private to ensure confidentiality. The first variable is the embedded .EXE file name. In this code, the payload executable file is set to 'song.exe'. The code only needs the file name because it can find the file in the resource folder inside the code. The next variable is the secret key byte to decrypt the payload. The code will use this key byte to decrypt the .EXE file when the file is ready to be executed.

```
private static final String ENCRYPTED_EXE_RESOURCE_NAME = "/song.exe";  
private static final byte[] KEY_BYTES = new byte[] {104, 90, -14, -127, -88, 13, 37, -15, -81, -29, -24, 33, 64, -6, 4, 0};
```

Figure 4. The variables in the dropper algorithm

This is the main method within the dropper algorithms, accountable for executing the core functionality. This method has many functions that play a major role for the payload to be executed successfully. The function of the method is to retrieve the embedded .EXE file as a stream from the resource folder. Based on the name in from the first variable, it will find the payload and retrieve it to be read by other code. Next, the method will read the stream as a byte to decrypt the file by using the second variable which is the secret key byte. The decryption process will be executed by other methods called the decryptBytes. The method will be invoked using Java's reflection. After the file is decrypted, another method will be called indirectly to create a temporary file to store the decrypted file. The temporary file is set to executable and will be executed when no errors appear. Both actions are handled by other methods and invoked indirectly.

3.4 Payload algorithms

In real malware, the payload usually carries out the malicious activity that compromises the victim's system. In this study, however, the payload was deliberately designed to demonstrate disruptive behavior without causing actual harm to the system or its data. The payload simulates such behavior by creating non-closable JFrames, playing background audio, and opening repeated browser tabs to overload the system's RAM. When the payload is executed, the first thing that will appear is a JFrame in the middle of the screen alongside with a song that will be played for 5 seconds. After that, many JFrames will be produced to fill up the screen while in the background, an infinite amount of browser tabs is opened making the device lags and unable to operate normally. The JFrame and browser tabs production are set to loop infinitely which means it will continue its function until the device crashes, or the user turns it off.

Most of the libraries imported in this code have a very straightforward and simple function as illustrated in Figure 5. The java.awt.* and javax.swing.* libraries manage GUI elements like JFrames. For sound playback, javax.sound.sampled.* works with java.io.InputStream to stream audio files. The java.net.URI library is responsible for launching browser tabs through URI links.

```
import java.awt.*;  
  
import javax.sound.sampled.*;  
import javax.swing.*;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.net.URI;  
import java.net.URISyntaxException;
```

Figure 5. The imported library in JAVA for the payload algorithm

As shown in Figure 6, the JFrame method places image-based windows randomly on the screen. These windows stay on top, are not resizable, and cannot be closed using the normal 'X' button.

```
public payloadRR(Dimension screenSize) {  
    super("YOU JUST GOR RICKROLLED");  
    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);  
    ImageIcon gifIcon = new ImageIcon(payloadRR.class.getResource("/rickroll.gif"));  
    JLabel gifLabel = new JLabel(gifIcon);  
    setAlwaysOnTop(true);  
    add(gifLabel, BorderLayout.CENTER);  
    pack();  
    setLocation(getRandomLocation(screenSize));  
    setResizable(false);  
    setVisible(true);  
}
```

Figure 6. The JFrame method in the payload algorithm

Figure 7 illustrates the main method of the algorithm. When the main method is executed, it first determines the screen size of the device. This information is useful for placing the JFrames within the visible area of the user's screen. The method then calls two slightly modified versions of previously defined functions: middletab() spawns a JFrame in the center of the screen, while playshorttrick() plays the same audio as the Playrick() method but limits it to five seconds.

After these initial actions, the method enters an infinite loop. In each iteration, it randomly spawns additional JFrames, opens browser tabs via the URI method, and plays audio using the Playrick() method. Importantly, Playrick() is executed in a separate thread to prevent the loop from waiting for the audio to finish before continuing, ensuring continuous disruption of the system.

```
public static void main (String args[]) {  
  
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();  
    middletab();  
    playshorttrick();  
  
    for(int i = 0; i < 5;){  
        new payloadRR(screenSize);  
        new payloadRR();  
  
        new Thread(() -> playrick()).start();  
    }  
}
```

Figure 7. The main method for dropper algorithm

4. RESULT AND DISCUSSION

This section presents the results and discussion of the proposed study.

4.1. ScanGuard scan results

After downloading the malware from the email link or transferred directly to the victim's device, using the ScanGuard feature called 'Smart Scan', it scans all the folders that exist in the virtual machine including the recently transferred malware. The Smart Scan failed to detect the malware on the victim's device as shown in Figure 8.

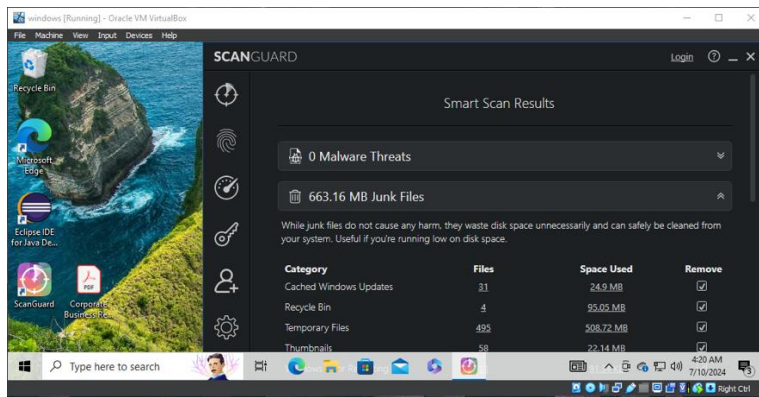


Figure 8. Result of Smart Scan by ScanGuard in the presence of malware

4.2. Windows security virus and threat protection scan results

Before the malware was activated, the task manager displayed typical resource usage for an idle virtual machine as illustrated in Figure 9. No major processes were running, and system performance appeared stable.

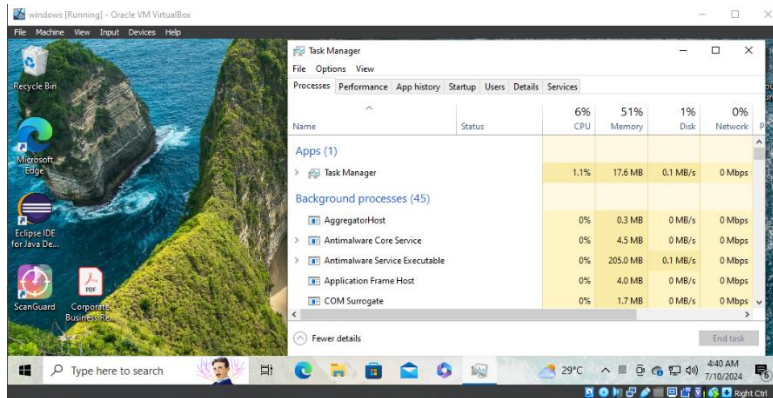


Figure 9. The CPU usage before the malware was activated

Once the malware ran, it quickly flooded the screen with JFrames, making it nearly impossible to interact with or monitor the virtual machine. Within seconds, the CPU usage spiked to 100%, as shown in Figure 10, and the system became sluggish and unresponsive.

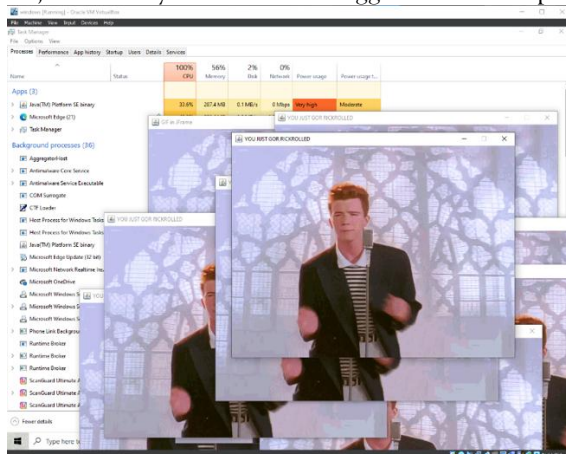


Figure 10. The CPU usage after the malware has been executed

5. CONCLUSION



This paper presents a practical and ethical approach to understanding malware behavior in Windows OS by developing a non-destructive prototype within a controlled virtual environment. Using the Agile development model, the study focused on building a dropper, payload, and evasion techniques that successfully bypassed antivirus detection and caused noticeable system disruption. While the results




highlight key vulnerabilities, there are limitations. The testing was confined to virtual machines, which may not fully replicate behavior on real hardware. The evaluation relied on a single antivirus tool, and the payload was kept simple to avoid real damage. Despite these constraints, the findings provide a strong foundation for safer malware research. Future work will focus on expanding antivirus testing, incorporating machine learning-based detection techniques like SVM and LSTM, and adapting the framework for other platforms such as Linux and macOS.

REFERENCES

- [1] K. H. Ukpabi and M. I. Abdullahi, Exploring Operating System Diversity: A Comparative Analysis of Windows, Mac OS, Android and IOS. *Journal of Systematic and Modern Science Research* (2024).
- [2] G. Agrawal, Accountability, Trust, and Transparency in AI Systems from the Perspective of Public Policy: Elevating Ethical Standards. *AI Healthcare Applications and Security, Ethical, and Legal Considerations*, pp. 148-162. IGI Global (2024).
- [3] B. G. Gordon, Vulnerability in Research: Basic Ethical Concepts and General Approach to Review. *Ochsner Journal*, pp. 34-38. 20, 1 (2020).
- [4] H. Shokouhinejad, R. Roozbeh, M. Hesamodin, R. Mahdi, A. Samuel, H. Griffin and A. G. Ali, Recent Advances in Malware Detection: Graph Learning and Explainability (2025).
- [5] Z. Yangchun, Z. Yuntian and Y. Ju, New Virus Infection Technology and Its Detection. 11th International Conference on Software Engineering and Service Science (ICSESS), pp. 388-394. IEEE (2020).
- [6] Y. Najih, The Effectiveness of Malware Mutation Techniques Against Anti-Malware Software. Master's thesis, Southeast Missouri State University, (2023).
- [7] J. Chatsomsanga and B. Chawalit, Malware Developing Guide: Encryption and Decryption. 24th International Conference on Advanced Communication Technology (ICACT), pp. 275-278. IEEE, (2022).
- [8] A. Sharma, M. Suman and I. Md Ruhul, An Experimental Analysis on Malware Detection in Executable Files using Machine Learning. 8th International Conference on Smart Computing and Communications (ICSCC), pp. 178-182. IEEE (2021).
- [9] J. Softić and V. Zanin, Windows 10 Operating System: Vulnerability Assessment and Exploitation. 21st International Symposium INFOTEH-JAHORINA (INFOTEH), pp. 1-5. IEEE (2022).
- [10] Vander-Pallen, A. Maame, A. Paul, I. Stuart and K. M. Tauheed, Survey on Types of Cyber Attacks on Operating System Vulnerabilities Since 2018 Onwards. *World AI IoT Congress (AllIoT)*, pp. 01-07. IEEE (2022).
- [11] A. Dalvi, K. Parth, K. Aditya and S. G. Bhirud, Dark Web Crawling for Cybersecurity: Insights into Vulnerabilities and Ransomware Discussions. 2nd International Conference for Innovation in Technology (INOCON), pp. 1-6. IEEE (2023).
- [12] P. Pa, M. Yin, T. Shunsuke, K. Tetsui, V. E. Michel, Y. Katsunari and M. Tsutomu, An Attacker's Dream? Exploring the Capabilities of ChatGPT for Developing Malware. 16th Cyber Security Experimentation and Test Workshop, pp. 10-18 (2023).
- [13] H. Yin, Detecting Mimicry Attacks in Windows Malware. (2023).
- [14] O. Shokunbi, U. Obinna, A. Damilare, A. Hannah, A. Oludele and A. Folasade, Security Integration in Agile Methodology. *IEEE SmartBlock4Africa*, pp. 1-10 (2024).
- [15] Uddin, A. Mirza, H. S. Sanjana, S.R. Mohammad and S. A. Mohammad, Deep Learning for Agile Malware Detection. *IEEE Region 10 Symposium (TENSYP)*, pp. 1-6. IEEE (2024).

Authors

	<p>Nazirah Abd Hamid is a senior lecturer in University Sultan Zainal Abidin, Terengganu, Malaysia. She holds a degree in Bachelor of Information Technology from University Utara Malaysia (UUM), in 2004, M. Sc. Com. (Information Security) from University Teknologi Malaysia (UTM), in 2011 and PhD in Computer Science from Universiti Teknikal Malaysia Melaka (UTeM), in 2023. Her research interests are Information Security, Cyber Security, Pattern Recognition and Data Mining. She can be contacted at email: nazirah@unisza.edu.my.</p>
	<p>Siti Dhalila Mohd Satar received her Ph.D. from Universiti Putra Malaysia in 2024 and her M.Sc. from Universiti Teknologi Malaysia in 2012. She is a Lecturer at Universiti Sultan Zainal Abidin, specializing in access control security systems, security services—including digital forensics, steganography, network security, and biometrics—and data security. Her research contributions focus on advancing security mechanisms to protect digital ecosystems.</p>

	<p>Ahmad Faisal Amri Abidin @ Bharun received his M. Sc. Com from Universiti Putra Malaysia in 2008. He is a senior lecturer specializing in Security Services (Digital Forensic, Steganography, Network Security, Public Key Infrastructure and Biometrics). He has almost 15 years of experience in cybersecurity. In addition, he has led various cybersecurity workshops.</p>
	<p>Mohd Fadzil Abdul Kadir received a Ph.D. in engineering (system engineering) from the Mie University, Mie, Japan, in 2012. Since 2006, he has been with the Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, where he is currently a Senior Lecturer. His main areas of research interest are digital image processing, pattern recognition, information security, and cryptography. He is also a member of the Malaysia Board of Technologists. He can be contacted at email: fadzil@unisza.edu.my.</p>
	<p>Mohamad Afendee Mohamed received his Ph.D. in Mathematical Cryptography from Universiti Putra Malaysia in 2011. Upon completion, he served the university for three years as a senior lecturer. In 2014, he moved to Universiti Sultan Zainal Abidin and later assumed an associate professor position. His current research interests include both theoretical and application issues in the domain of data security and mobile and wireless networking. He has authored more than 100 articles that have appeared in various journals, book chapters, and conference proceedings. He can be contacted at email: mafendee@unisza.edu.my.</p>