A Multi-Layer, Explainable, Rule-Based Threat-Detection System for Enterprise and Branch-Office Environments

Master of Science in Cybersecurity and Digital Forensics

MD ABDULLAH AL MAMUN

Student ID Number: 2434327

University of Wales Trinity Saint David, London

Supervisor: Dr Amit Naik PhD, MSc, BEng. MBCS

Lecturer in Computing

Submission: 25 of September 2026

DECLARATION

I, **MD. ABDULLAH AL MAMUN**, declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed: **MD. ABDULLAH AL MAMUN**

Date: 25/09/2026

FORM OF CONSENT

I, **MD. ABDULLAH AL MAMUN**, hereby consent that my Project, submitted in candidature for the M.Sc. in Cybersecurity and Digital Forensics, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed: **MD. ABDULLAH AL MAMUN**

Date: 25/09/2026

Table of Contents

Abstract 6

CHAPTER 1 — INTRODUCTION 7

1.1 Background and Context 7

1.2 Problem Statement 8

CHAPTER 2 — LITERATURE REVIEW 11

2.1 Introduction to the Review 11

2.2 Global Data Protection Regulatory Landscape 11

2.3 ISO/IEC 27001:2022 as a Jurisdiction-Agnostic Technical Backbone 13

2.4 The Phishing Threat Landscape and the Efficacy of Detection Modalities 16

2.4.1 The Limitations of Machine Learning in Enterprise Phishing Detection 16

2.4.2 The Case for Explainable, Rule-Based Heuristic Detection 18

2.4.3 The Emerging Middle Ground: Explainable Machine Learning 19

2.5 Advanced Obfuscation Vectors: Malicious Documents and "Quishing" 20

2.5.1 Static Structural Analysis of Office Documents and PDFs 20

2.5.2 Extracting Hidden Links and the Rise of QR-Code Phishing ("Quishing") 21

2.6 Vulnerability Assessment and Port Scanning within the Branch Office Context 22

CHAPTER 3 — RESEARCH METHODOLOGY 25

3.1 Introduction to the Methodological Framework 25

3.2 Research Philosophy: The Pragmatist Paradigm 25

3.3 Research Approach: A Mixed Deductive-Inductive Strategy 27

3.3.1 The Deductive Phase: Theory to Implementation 27

3.3.2 The Inductive Phase: Empirical Observation to Algorithmic Refinement 28

3.4 Research Strategy: Design Science Research (DSR) 28
3.5 Data Generation and Collection Methods 30
3.5.1 Structured Synthetic Data Generation 30
3.5.2 Live Operational Data Validation 32
3.6 Evaluation and Analysis Framework 33
3.6.1 Quantitative Metric Evaluation 33
3.6.2 Qualitative Diagnostics and Root-Cause Analysis 33
3.6.3 Cross-Language Consistency Verification 34
3.7 Ethical Considerations and Data Privacy 34
3.8 Limitations of the Chosen Methodology 35
CHAPTER 4 — Design Process 37
4.1 Introduction to the System Design Paradigm 37
4.2 The "One Engine, Many Doors" Architectural Philosophy 37
4.3 System Architecture and Data Flow 39
4.4 Heuristic Rule Design and Threshold Calibration 41
4.5 Cross-Environment Portability: The Python-JavaScript Bridge 44
4.6 Test Planning and Synthetic Evaluation Design 45
CHAPTER 5 — IMPLEMENTATION 47
5.1 Introduction to the Implementation Phase 47
5.2 Module 1: Core File Scanner (core_scanner.py) 47
5.3 Module 2: Email Scanner (email_scanner.py) 48
5.4 Module 3: Live Mailbox Scanner (live_mailbox_scanner.py) 49
5.5 Module 4: Client-Side Browser Extension 49
CHAPTER 6 — EVALUATION OF RESULTS 51
6.1 Introduction to the Evaluation Phase 51
6.2 Evaluation Methodology and Synthetic Test Environment 51

6.3 Case Study 1: PowerPoint Metadata and Macro False-Positive Resolution 52

6.4 Case Study 2: Live-Inbox False-Positive Diagnostics (The Query-String '@' Symbol Anomaly) 53

6.5 Case Study 3: Browser Extension Content Security Policy (CSP) Interception Challenge 54

6.6 Cross-Engine Consistency Verification (Python vs. JavaScript Parity)55

6.7 Comprehensive Regression Testing and Final System State 56

6.8 Positioning Against Literature-Reported Benchmarks 56

6.9 A Threats-to-Validity Perspective on the Evaluation 57

6.10 Tracing the Rule Engine Against Documented Real-World Cases ... 58

CHAPTER 7 — CONCLUSIONS, RECOMMENDATIONS, AND REFLECTION 60

7.1 Conclusions 60

7.2 Technical and Operational Limitations 61

7.3 Recommendations for Future Work 62

7.4 Personal Reflection 63

References……..65

Abstract

Multinational corporations exchange large volumes of files and email every day, and phishing links together with malicious macro-enabled documents remain among the most common ways attackers gain initial access. Many existing single-layer defences give little explanation for their decisions, which limits how much a non-specialist administrator can trust or act on their output. This project designs, implements, and evaluates a multi-layer, explainable threat-detection system that applies one consistent, rule-based detection policy across four points of exposure: files stored locally, email attachments and links (both offline demonstration data and a live connected inbox), and web browsing. An 8-rule heuristic scoring engine, covering indicators such as IP-based URLs, brand-name spoofing, and links hidden behind images or QR codes, was implemented in Python and ported identically into JavaScript for a browser extension, and formally verified to produce matching results across both implementations. The system was tested against a structured set of synthetic files and emails, and further validated against a real, live Gmail inbox, during which testing surfaced and resolved three logic-level issues, including a false positive that only became visible once real, unpredictable email traffic was examined rather than controlled test data alone. The results show that a lightweight, fully explainable heuristic approach can deliver consistent and verifiable detection across multiple delivery channels without the training-data and interpretability limitations associated with machine-learning-based alternatives, while also highlighting practical deployment considerations relevant to real-world adoption in an MNC environment.

Keywords: phishing detection; heuristic/rule-based security; data loss prevention; macro malware; ISO/IEC 27001; browser extension security; multi-layer architecture.

CHAPTER 1 — INTRODUCTION

1.1 Background and Context

Modern businesses depend on the constant, fast movement of digital information. Multinational corporations (MNCs) operate across large, interconnected networks. Every day, they exchange enormous volumes of email, financial records, and documents. PDFs, Word files, PowerPoint decks, spreadsheets, and images are not just incidental to this process, they are how the business runs, moving continuously between staff, clients, partners, branch offices, and outside vendors.

This constant exchange also creates a large attack surface. As organizations move more of their operations online, they become more attractive targets for cybercriminals, ranging from opportunistic attackers to well-resourced, state-linked groups. Most of these attackers do not rely on breaking through a firewall directly. Instead, they rely on tricking a person, using a file or a link that looks ordinary but is not.

Common examples of this include Word or Excel files with hidden VBA macros that run automatically once a user clicks "Enable Content" [1]. I have also seen PDFs with embedded JavaScript or an auto-launch action that fires the moment the file opens, and links hidden behind an image or encoded inside a QR code rather than written out as visible text [2]. Alongside these methods, phishing emails continue to imitate trusted brands, banks, and internal company communications convincingly enough to catch even experienced users off guard.

Industry reporting consistently backs this up. For example, the FBI's Internet Crime Complaint Centre (IC3) continues to report phishing and business email compromise as among the most damaging and frequently reported categories of cybercrime worldwide [3]. Also, recent systematic reviews of the threat landscape further highlight the persistent adaptability of these phishing attacks against standard defences [4, 5].

Despite this reality, many organizations still rely on single-layer defences, like a standalone antivirus tool or a basic spam filter, that behave like a black box.

They block or allow something but rarely explain why they made that decision. For a small or mid-sized branch office without a dedicated security team, this creates a real problem. When a legitimate document gets blocked with no explanation, an already-stretched IT administrator is left to either trust the tool blindly or spend valuable time manually checking it themselves. Neither option is good, and this exact opacity makes it harder to keep a clear audit trail for compliance purposes.

This sits against a regulatory backdrop, including UK GDPR, EU GDPR, and international standards such as ISO/IEC 27001:2022 [6, 7], that requires organizations to apply “appropriate technical and organizational measures” to protect data, without specifying exactly what those measures should look like in practice. Implementing these frameworks in a practical IT environment remains a well-documented challenge [8]. I view that gap as both a compliance challenge and, from an engineering point of view, a clear opportunity. It leaves room for a lightweight, explainable tool like the one I built in this project to fill a real, practical need rather than just satisfy a checkbox.

1.2 Problem Statement

While looking into current business environments, I identified a critical vulnerability: the heavy reliance of small-to-mid-sized multinational branch offices on generic, vendor-locked commercial antivirus and endpoint detection solutions. Because these regional branch offices frequently operate without dedicated, 24/7 Security Operations Centres (SOCs) or highly specialized cybersecurity personnel, local IT administrators depend heavily on off-the-shelf security utilities. The main issue here is that these tools typically offer minimal insight into the underlying heuristic mechanics or threat intelligence indicators driving their automated security flags.

Another issue is that commercial security suites often enforce separate security policies across different operational entry points within the same organization. For example, a corporate email security gateway might successfully block a malicious file or phishing link. However, that exact same file might successfully execute if a user transfers it via a local USB flash drive, downloads it via a cloud storage synchronization client, or accesses it through

active web browsing on an unprotected network segment. This policy inconsistency stems directly from the historical absence of a unified, multi-entry-point protection architecture that applies a single, cohesive set of detection rules across the entire enterprise ecosystem.

Adding to this architectural fragmentation is an important gap I noticed in current academic and industrial phishing-detection literature. A review of existing cybersecurity research shows that many studies focus mainly on URL text or email body content. I found fewer integrated detection frameworks that examine email body text, complex attachment structures, hidden hyperlinks in images, and matrix-barcode payloads such as QR-code-based "quishing" within one processing pipeline [9]. Most existing commercial and academic systems treat email scanning, local file inspection, and web URL filtering as separate engineering areas. This can lead to duplicated code, higher processing overhead, and different security policies. Das Guptta et al. [10] confirm that hyperlink-derived features can provide useful phishing signals without relying on third-party services, which supports the approach used by this project's Office and PDF parsers.

The widespread opacity of modern threat detection tools also creates systemic friction between automated security controls and human operators. When an administrative officer cannot inspect the specific rule, regular expression, or heuristic indicator that triggered a security flag, auditability completely breaks down. Under strict regulatory standards such as ISO/IEC 27001, organizations must be able to demonstrate clearly how they evaluate, document, and manage information security risks. While modern black-box machine learning (ML) and artificial intelligence (AI) models have achieved high nominal accuracy in controlled, synthetic experimental settings, they mainly worsen this transparency deficit in practical deployment. The complex internal weight distributions, hidden layers, and inherent mathematical opacity of deep neural networks make their decision-making processes exceptionally difficult to audit or explain to non-technical stakeholders. On top of that, these models are notoriously susceptible to adversarial perturbations, minor, intentional alterations to input data designed to evade detection [11, 12], making them poorly suited for non-specialist administrative environments where

explainability and deterministic reliability are absolute operational prerequisites.

For this reason, the central problem I am addressing through this detailed research project is the historical absence of a lightweight, easy to explain, multi-entry-point technical control that uniformly applies a consistent, careful detection policy across diverse threat delivery vectors. Specifically, there is an urgent operational need for a system that can simultaneously monitor static desktop files, offline email archives, live organizational email streams, and real-time web browser navigation, all while providing transparent, deterministic, and human-readable rationales for every automated security verdict it renders.

CHAPTER 2 — LITERATURE REVIEW

2.1 Introduction to the Review

This chapter provides a detailed, critical evaluation of the existing academic literature, industrial cybersecurity frameworks, and international regulatory standards pertinent to enterprise threat detection. It seeks to contextualize the specific engineering decisions made within this project, most the adoption of a unified, rule-based heuristic architecture over decentralized machine-learning models. The review in a structured way explores the global landscape of data protection regulations, analyses the jurisdiction-agnostic technical controls mandated by ISO/IEC 27001, critically compares the performance in practice of machine-learning versus explainable heuristic systems, and evaluates the specific mechanics of static structural analysis for macro and active-content detection. By synthesizing these diverse knowledge domains, this chapter establishes the theoretical foundation upon which the project's "one engine, many doors" architecture is built, whilst clearly identifying the existing literature gaps that this research aims to bridge.

2.2 Global Data Protection Regulatory Landscape

The global regulatory environment governing data privacy and information security has experienced a rapid evolution over the past decade. This paradigm shift was largely catalysed by a series of high-profile, catastrophic corporate data breaches that exposed the systemic vulnerabilities inherent in global data supply chains. As a result, sovereign states and supranational entities have aggressively pursued legislative frameworks designed to enforce corporate accountability, mandate strict security controls, and protect individual data subjects. However, despite this shared overarching objective, the resulting legal frameworks exhibit considerable variations in their territorial scope, enforcement mechanisms, operational definitions of personal data, and foundational legal philosophies.

Academic comparative regulatory analyses frequently identify three distinct philosophical models dominating the current global data protection landscape: the rights-based European model, the consumer-control model prevalent in

the United States, and the state-centric models emerging in Asian jurisdictions [6].

The European Union's General Data Protection Regulation (EU GDPR), implemented in 2018 and subsequently mirrored by the United Kingdom (UK GDPR) following Brexit, represents a well-established rights-based regulatory framework. This regime is based on the principle that data privacy is a fundamental human right. As a result, the GDPR imposes strict requirements for explicit, informed consent before data processing, requires Data Protection Officers (DPOs) in relevant organisations, and enforces the principle of "Data Protection by Design and by Default" (Article 25). From an enterprise security perspective, Article 32 of the GDPR is particularly relevant. It requires organisations to use "appropriate technical and organisational measures to ensure a level of security appropriate to the risk." Importantly, the regulation remains technologically neutral and does not prescribe a particular software solution or architecture. Organisations therefore need to show that their technical controls, such as the multi-layer file and email scanning system developed in this project, are appropriate to their threat model. The GDPR can also impose administrative fines of up to 4% of a company's total worldwide annual turnover, making data protection an important issue for corporate governance as well as IT security.

Conversely, the data protection landscape in the United States remains heavily fragmented, lacking a singular, detailed federal privacy law equivalent to the GDPR. Instead, the US relies on a patchwork of sector-specific regulations (such as HIPAA for healthcare and GLBA for financial services) and emerging state-level legislation. The California Consumer Privacy Act (CCPA), significantly amended by the California Privacy Rights Act (CPRA), represents the strictest state-level framework to date. Unlike the GDPR's proactive, consent-driven ("opt-in") model, the CCPA operates primarily on a reactive, consumer-control ("opt-out") philosophy, focusing heavily on a consumer's right to know what data is being collected and the right to prohibit its sale. However, like the GDPR, the CCPA imposes liability on businesses that fail to maintain "reasonable security procedures and practices." Academic literature comparing these regimes notes that while the CCPA is narrower in

philosophical scope, the technical security controls required to satisfy its "reasonable security" mandate are largely indistinguishable from those required by the GDPR.

Expanding the comparative analysis to the Asia-Pacific region reveals alternative regulatory paradigms. China's Personal Information Protection Law (PIPL), enacted in 2021, integrates elements of the GDPR (such as individual rights and consent) with a distinct state-centric philosophy prioritizing national security and careful controls over cross-border data transfers. Similarly, frameworks like Japan's Act on the Protection of Personal Information (APPI) and South Korea's Personal Information Protection Act (PIPA) represent hybrid approaches, attempting to balance strong individual privacy rights with business-friendly enforcement mechanisms designed to help digital economic growth.

Navigating this complex, often contradictory web of global regulations presents an immense operational challenge for multinational corporations. An enterprise operating simultaneously in London, California, and Tokyo cannot effectively manage cybersecurity if it attempts to build bespoke technical infrastructures for every distinct jurisdiction. This reality dictates a fundamental operational necessity: MNCs must adopt universally applicable technical security controls that satisfy the highest common regulatory denominator. In current compliance practice, systems engineered to satisfy the careful technical requirements of the GDPR and UK GDPR are widely considered capable of simultaneously satisfying the less prescriptive security mandates of frameworks like the CCPA and APPI. For this reason, while this project utilizes the UK GDPR as its primary illustrative legal context, the technical threat detection system it proposes is clearly designed to address the universal legal requirement for "appropriate technical security measures" shared across all major global regulatory regimes.

2.3 ISO/IEC 27001:2022 as a Jurisdiction-Agnostic Technical Backbone

Unlike the fragmented and localized nature of national data protection legislation, the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) provide globally recognized,

jurisdiction-agnostic frameworks for information security. ISO/IEC 27001 is the world's most prominent standard for establishing, implementing, maintaining, and continually improving an Information Security Management System (ISMS). For multinational corporations, ISO/IEC 27001 functions effectively as a critical "translation layer" between abstract legal mandates (such as the GDPR's "appropriate measures") and concrete, auditable engineering practices.

The most recent detailed revision of the standard, ISO/IEC 27001:2022, introduced important structural and thematic changes designed specifically to address the current enterprise threat landscape [7, 8]. A primary contribution of this revision was the complete restructuring of Annex A, which transitioned from 114 controls categorized across 14 domains (in the 2013 edition) to a streamlined set of 93 controls categorized into four distinct themes: Organizational (37 controls), People (8 controls), Physical (14 controls), and Technological (34 controls). This thematic restructuring is highly relevant to applied software engineering projects, as it clearly delineates where automated software systems fit within the broader enterprise security posture.

Another issue is that the 2022 revision introduced 11 entirely new controls designed to mitigate emerging threat vectors that were inadequately addressed in previous iterations. Several of these newly mandated technological controls map directly and clearly onto the system architecture developed within this research project:

- **Control A.5.7 (Threat Intelligence): This control mandates that organizations gather and analyse information regarding specific threats to mitigate risks proactively. The heuristic scoring engine built in this project, which encodes known attacker behaviours such as the abuse of specific top-level domains (.tk,.xyz), the use of specific URL shorteners, and brand impersonation tactics, acts as an put into practice execution of actionable threat intelligence.**
- **Control A.8.10 (Information Deletion): This control requires that information stored on IT systems be securely deleted when no longer required. The Python-based email parsing module**

developed in this study clearly adheres to this control by using temp-file structures and strong shutil. run tree cleanup routines within finally execution blocks. This privacy-by-design mechanism ensures that potentially sensitive email attachments extracted for security scanning are clearly purged from disk immediately upon scan completion, regardless of the scan's outcome or unexpected software exceptions.

- **Control A.8.12 (Data Leakage Prevention):** This control requires technical measures to monitor, detect, and prevent the unauthorized disclosure of sensitive data. While this project focuses primarily on ingress threat detection (preventing malware from entering), the underlying architectural capability to intercept specific file types and evaluate URLs in real-time forms the foundational prerequisite for any DLP system.
- **Control A.8.23 (Web Filtering):** This control mandates that access to external websites must be managed to reduce exposure to malicious content. The Chromium-based browser extension developed in this project directly implements this control, shifting the filtering capability from a centralized perimeter gateway directly to the client endpoint, effectively intercepting navigation to empirically verified phishing domains before the browser can resolve the connection.

Also, the academic literature highlights the growing adoption of ISO/IEC 27701, a privacy extension to ISO/IEC 27001 that outlines the requirements for a Privacy Information Management System (PIMS). Industry analysts observe that MNCs increasingly use ISO/IEC 27701 to address overlapping regulatory requirements, such as the GDPR and CCPA, through a single set of auditable controls. Because ISO/IEC 27001 and 27701 are not tied to one legal jurisdiction, they provide a widely applicable reference point for enterprise system design. When an organisation can show an auditor that its automated file and email scanning controls meet the relevant Annex A requirements, compliance reviews can be more straightforward. For this reason, this project uses ISO/IEC 27001:2022 as a main technical reference,

linking the security controls used in the system with wider compliance requirements.

2.4 The Phishing Threat Landscape and the Efficacy of Detection Modalities

Phishing, historically defined as the fraudulent attempt to obtain sensitive information by disguising oneself as a trustworthy entity in electronic communication, has evolved from a blunt, high-volume spam operation into a highly targeted, sophisticated, and technologically complex attack vector. The modern phishing landscape encompasses diverse delivery mechanisms, including traditional email body URLs, malicious document attachments engineered to execute payloads upon interaction, and increasingly prevalent tactics such as "spear-phishing" (highly targeted attacks using extensive social engineering) and "quishing" (the use of malicious QR codes to bypass text-based optical character recognition scanners).

Within the academic discourse regarding automated phishing detection, two primary methodological paradigms dominate: Machine Learning (ML)/Artificial Intelligence (AI) models and Rule-Based/Heuristic systems. The intense debate surrounding the comparative operational efficacy of these two approaches is central to the architectural decisions made in this research project.

2.4.1 The Limitations of Machine Learning in Enterprise Phishing Detection

Machine-learning models, encompassing algorithms such as Random Forests, Support Vector Machines (SVMs), and deep learning architectures like Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, are extensively researched within current academic cybersecurity literature. When evaluated in isolated, strictly controlled experimental settings using historical, statically labelled datasets, these ML classifiers frequently report exceedingly high nominal accuracy rates, often exceeding 98%. Recent examples include [13], an XGBoost-based browser extension, and [14], reporting 98.32% accuracy and outperforming Google Safe Browsing over a 15-day zero-day window. However, a critical review of

the literature reveals severe, often difficult to overcome practical challenges when attempting to deploy these models into dynamic, unpredictable enterprise production environments.

The primary systemic vulnerability of ML-based phishing detection is its inherent opacity. Deep learning models operate as mathematical "black boxes." While the model provides a probabilistic classification (e.g., "99% probability of phishing"), the internal decision-making pathway, the specific distribution of weights across hidden neural layers that led to that conclusion, is exceptionally difficult, if not impossible, for human analysts to decipher. This opacity creates large operational friction in enterprise environments. When an ML-driven security gateway quarantines a highly sensitive corporate contract or blocks a critical partner communication, it cannot provide the system administrator with a specific, human-readable rationale. The administrator cannot determine if the email was blocked due to a suspicious domain name, anomalous textual sentiment, or an obscure metadata tag. As a result, the administrator cannot confidently authorize an override, adjust a firewall rule, or document the rationale for an audit, violating the core transparency requirements of frameworks like ISO/IEC 27001. Also, as demonstrated by Siriwardhana et al. [15], while models like XGBoost can be integrated into browser extensions for real-time detection, they still face important practical hurdles in dynamic enterprise environments compared to transparent rule-based systems.

Also, a landmark IEEE study [11] investigating the reliability and robustness of ML-based phishing URL detectors highlighted their severe susceptibility to adversarial perturbations. Sabir et al. [12] corroborate this at scale, evaluating fifty ML-based phishing URL detectors and finding adversarial training yields only marginal robustness gains. Attackers routinely employ evasion tactics designed clearly to confuse the statistical models, such as slightly modifying the visual structure of a webpage, using novel URL obfuscation techniques absent from the model's original training data, or injecting invisible, benign text into an email to alter its statistical payload. The IEEE study demonstrated that even state-of-the-art deep learning classifiers experience large performance degradation when confronted with these novel adversarial inputs, leading to

unacceptable rates of false negatives (allowing malware to pass) and false positives (blocking legitimate traffic).

Finally, maintaining an effective ML model requires a very large, continuous investment in data infrastructure. Threat landscapes evolve daily; For this reason, ML models demand continuous, resource-intensive retraining using newly acquired, carefully labelled datasets to prevent model drift. For small-to-mid-sized multinational branch offices lacking dedicated data science teams and vast computational resources, deploying and maintaining bespoke ML security models is practically and economically unfeasible. Systematic reviews of deep-learning phishing-email detection reach the same conclusion: [4] and [5] both identify adaptability to novel phishing behaviour, not raw accuracy, as the field's unresolved challenge - a limitation this project's training-data-free engine sidesteps entirely.

2.4.2 The Case for Explainable, Rule-Based Heuristic Detection

Conversely, rule-based or heuristic detection systems operate on mainly different principles. Rather than relying on large datasets to identify obscure statistical correlations, heuristic engines utilize a concise, transparent set of deterministic rules engineered by human domain experts. These rules analyse specific, identifiable structural and semantic indicators known to be strongly associated with malicious intent. Examples include analysing a URL to see if it utilizes a raw IPv4 address instead of a registered domain name (a classic indicator of ephemeral phishing infrastructure), detecting the presence of anomalous top-level domains (.xyz,.tk) with high statistical abuse rates, identifying the presence of homograph attacks via Punycode conversion, and detecting structural brand impersonation (e.g., paypal-security-update.com instead of paypal.com).

The academic literature consistently demonstrates that lightweight heuristic systems, when carefully tailored for specific operational environments, can achieve strong detection accuracy without the computational burden or opacity of ML models. For example, SatheeshKumar et al. [16] show that a lightweight, non-machine-learning heuristic engine can achieve detection rates exceeding 90% while executing in milliseconds and demanding virtually zero CPU

overhead, making such approaches ideal for deployment in edge environments such as browser extensions or lightweight local branch-office servers. Tupsamudre et al. [17] corroborate this with PhishMatch, and recent studies by Phoon et al. [18] and Kala et al. [19] further emphasize the necessity of multi-layered detection strategies. This reinforces the argument that layered, rule-based architectures provide strong defence mechanisms without the overhead of complex ML models.

Importantly, heuristic detection directly and clearly solves the enterprise requirement for explainability. Because a heuristic engine executes deterministic logic (IF Condition X is met, THEN add Y points to the risk score), it can always output an exact, human-readable rationale alongside its verdict. If a file is blocked, the heuristic system clearly states: Blocked because the URL 'http://192.168.1.5' utilizes a raw IP address (Score +35) and attempts to impersonate the brand 'Microsoft' (Score +30). This absolute transparency empowers IT administrators to audit system decisions rapidly, confidently resolve false positives by tuning specific rules, and maintain clear, compliant audit logs.http://192.168.1.5**Error! Hyperlink reference not valid.**

For this reason, this project clearly rejects the ML paradigm in favor of constructing a transparent, 8-rule heuristic scoring engine. This architectural decision is deeply grounded in the literature: it avoids the intensive training-data requirements and adversarial vulnerabilities identified in ML studies, while providing the deterministic explainability that modern enterprise governance and compliance frameworks demand.

2.4.3 The Emerging Middle Ground: Explainable Machine Learning

A growing body of recent literature applies Explainable AI (SHAP) directly to phishing detection, targeting the same transparency deficit discussed above. [20] show a SHAP-instrumented XGBoost classifier reaching 97.78% accuracy while still producing human-readable feature attributions, and [21] find that SHAP-derived feature importance shifts substantially when a model is deployed outside its training distribution. A parallel strand targets email specifically: [22] reframe spear-phishing detection as identity fact-checking against a knowledge base, deployable as a mail-client plugin.

This narrows the gap between the two paradigms but does not remove it for this project's context. SHAP explanations are post-hoc statistical attributions rather than deterministic rules, remain unstable across datasets, and still require the training-data and retraining pipeline this project set out to avoid. The heuristic approach here is therefore a context-specific trade-off, not a rejection of explainable ML in principle.

2.5 Advanced Obfuscation Vectors: Malicious Documents and "Quishing"

A systematic review of current phishing-email detection techniques reveals a critical, prevalent structural deficit in existing systems [9]: the vast majority of commercial tools and academic models rely almost entirely on the isolated analysis of visible URL text strings or the natural language processing of the email body content. There is a profound scarcity of unified systems capable of simultaneously evaluating the text, the underlying structural intent of attached files, and obscured payload delivery mechanisms. This project specifically targets this literature gap by integrating advanced structural analysis.

2.5.1 Static Structural Analysis of Office Documents and PDFs

The embedding of malicious Visual Basic for Applications (VBA) macros within Microsoft Office documents (Word, Excel, PowerPoint) remains one of the most resilient and devastating malware delivery mechanisms in existence. Macros exploit the legitimate software automation features provided by Microsoft to execute arbitrary PowerShell commands, download secondary payloads (such as ransomware or Remote Access Trojans), and establish persistent footholds on local machines. Because macros leverage legitimate software functionality rather than exploiting memory corruption vulnerabilities, traditional signature-based antivirus solutions are notoriously unreliable at detecting novel, heavily obfuscated macro code.

The industry standard approach to detecting advanced macros involves dynamic behavioural sandboxing—executing the suspicious document within an isolated virtual machine and monitoring its behaviour for malicious activity. However, sandboxing is computationally expensive, introduces significant latency (often delaying email delivery by several minutes), and is highly

susceptible to evasion. Sophisticated malware samples are routinely engineered with environmental awareness; they can detect the presence of virtualization software (such as VMware or VirtualBox), check the number of CPU cores, or monitor mouse movement. If the malware detects it is running within a sandbox, it remains entirely dormant, executing its malicious payload only after it determines it has safely reached a legitimate user's workstation.

To circumvent the computational limitations and evasion vulnerabilities of sandboxing, this project relies on the academic principles of static structural analysis. Modern Microsoft Office files (the Office Open XML format, such as .docx and .pptx) do not function as monolithic binary blobs; they are structurally ZIP archives containing a complex hierarchy of internal XML files and relationship directories. According to the strict structural definitions of the format, a macro-enabled file must contain a specific internal binary file named vbaProject.bin. If this file is absent from the internal ZIP directory, it is mathematically impossible for the document to execute VBA macro code.

A comprehensive 2023 academic study [1] focusing on the detection of malicious Office macros demonstrated the empirical efficacy of utilizing static document structural features as a rapid, computationally efficient preliminary filter. By utilizing Python’s built-in zip file module to statically inspect the internal archive listing for the vbaProject.bin marker—without ever executing the file or relying on a sandbox—this project achieves rapid, zero-dependency macro detection suitable for low-resource branch office deployments.

Similarly, Portable Document Format (PDF) files are frequently weaponized by attackers who embed malicious JavaScript or manipulate the document's internal /OpenAction trigger objects to execute commands silently upon opening. Consistent with the broader literature on malicious document detection, this project applies static structural analysis to PDFs, directly traversing the internal object hierarchy to detect the presence of /Names -> /JavaScript dictionaries and /OpenAction directives without requiring the computational overhead of rendering the PDF via a viewing engine.

2.5.2 Extracting Hidden Links and the Rise of QR-Code Phishing ("Quishing")

A frequently overlooked yet highly effective obfuscation vector involves the use of invisible or disguised hyperlinks. Attackers routinely embed malicious URLs not as visible text but superimposed invisibly over entirely benign-looking images within Word documents, PowerPoint presentations, or PDF files. An employee clicking anywhere on a seemingly innocuous "company logo" image might unwittingly trigger a navigation event to a credential-harvesting server. Traditional security scanners that rely on text extraction APIs typically miss these links entirely, as they are stored within the document's relationship metadata tables rather than the visible text runs. This project addresses this specific gap identified in the literature by engineering custom parsers that sweep the internal relationship XML files (doc.part.rels in Word) and hyperlink annotations (/Annots arrays in PDFs) to extract and penalize hidden URLs.

Furthermore, the rapid post-pandemic normalization of Quick Response (QR) codes has birthed a novel phishing methodology termed "Quishing" (QR-phishing). Attackers embed malicious URLs inside QR codes and place these codes onto legitimate-looking PDF invoices or PNG image attachments, urging the user to scan the code with their mobile device to "authorize a payment" or "verify an account." Because the malicious URL is mathematically encoded as a matrix barcode rather than a text string, traditional email gateways and text-based URL scanners are entirely blind to the payload, allowing the malicious email to bypass security controls effortlessly. The systematic reviews cited earlier identified QR-code-based obfuscation as a significant, heavily under-addressed gap in enterprise security tooling, a gap directly targeted by recent malicious-URL-focused QR scanning research [23]. [2] situates this within the post-pandemic surge in QR-code adoption, noting countermeasures remain fragmented rather than integrated into general-purpose detection pipelines. This project directly confronts this challenge by integrating the pyzbar machine-vision library into its image analysis module, actively decoding any QR codes detected within email attachments or desktop images, extracting the hidden URL payload, and forcing it through the identical 8-rule heuristic scoring engine applied to standard text links.

2.6 Vulnerability Assessment and Port Scanning within the Branch Office Context

While the primary focus of this research centres on file, email, and web-based threat detection, a detailed enterprise security posture also necessitates careful internal network visibility. Vulnerability assessment is traditionally categorized into three distinct operational phases: network discovery, port scanning, and web-application security auditing. Port scanning, the systematic probing of a server or host for open transmission ports, is the foundational mechanism for identifying unauthorized services, shadow IT infrastructure, and potential network entry points prior to their exploitation by malicious actors. In the broader context of web and application security, researchers like Sirmulla and Manickam [24] highlight the importance of dedicated detection mechanisms for specific threats like SQL injection, which underscores the need for detailed internal scanning alongside file and email protection.

A 2024 comparative academic analysis of heavyweight commercial vulnerability scanners (including industry standards such as Nmap, Nessus, and OpenVAS) confirmed their efficacy in traditional, centralized corporate network topologies [25]. However, the study also critically highlighted their operational limitations, noting that their large licensing costs, complex configuration requirements, and considerable network overhead make them poorly suited for deployment in small, dynamic, or resource-constrained regional branch offices.

This perspective is strongly reinforced by a 2021 study that advocates repositioning port and vulnerability scanning not merely as an aggressive reconnaissance tool for penetration testers, but as a crucial, continuous defensive mechanism for local network administrators to maintain an accurate, real-time inventory of exposed internal services [26]. The literature detailing the development of custom, lightweight Python-based port scanners provide strong academic justification for engineering bespoke network visibility tools. For environments like small-to-mid-sized MNC branch offices, a lightweight, custom-built scanning script that rapidly enumerates open ports without the heavy dependency footprint of commercial scanners provides immense defensive value.

The standard operational workflow for network vulnerability assessment, moving in a structured way from target host identification, to port service enumeration, and finally to vulnerability signature matching, offers a strong structural template that deeply informed the initial conceptualization of the security architecture in this project. Although strict project time constraints necessitated that the specific network-vulnerability-assessment module remain unimplemented as a functional component within the final software deliverable (it is thoroughly documented as a primary avenue for future work in Chapter 7), the theoretical principles of lightweight, decentralized security monitoring derived from this literature directly influenced the core design philosophy of the implemented file and email scanning modules.

CHAPTER 3 — RESEARCH METHODOLOGY

3.1 Introduction to the Methodological Framework

The establishment of a careful methodological framework is the foundational prerequisite for any applied computer science research, particularly within the specialized domain of enterprise cybersecurity. A methodology serves not merely as a chronological procedural log, but as the theoretical and philosophical justification for how knowledge is generated, tested, and validated within the scope of the study. This chapter in a structured way delineates the research philosophy, approaches, strategies, and data collection mechanisms employed to evaluate the multi-layer; heuristic-based threat detection system developed in this project.

The primary objective of this research is to engineer and empirically validate a functional software artefact capable of correctly differentiating between benign operational digital assets and maliciously engineered threats across four distinct enterprise entry points. Given this applied, engineering-centric objective, traditional social science methodologies, which frequently rely on human participant surveys, observational behavioural case studies, or purely theoretical mathematical modelling, are structurally insufficient. Instead, this chapter outlines a methodology heavily rooted in Design Science Research (DSR), driven by a pragmatist philosophy, and using a mixed deductive-inductive approach to system refinement. This framework guarantees that the resulting empirical data regarding detection accuracy, heuristic scoring reliability, and false-positive occurrences is objectively verifiable, highly reproducible, and directly relevant to the operational realities of modern multinational corporations (MNCs).

3.2 Research Philosophy: The Pragmatist Paradigm

In academic research, the underlying philosophy dictates the researcher's ontological (the nature of reality) and epistemological (how we acquire valid knowledge) assumptions. in evaluating cybersecurity architectures, researchers frequently must navigate between two extreme philosophical paradigms: pure positivism and interpretivism.

Positivism asserts that reality is singular, objective, and entirely independent of the researcher, arguing that knowledge can only be acquired through strict, mathematical quantification and observable physical phenomena. While a positivist stance aligns well with the purely quantitative metrics of this project, such as measuring precise execution times, calculating strict false-positive percentages, or verifying deterministic Boolean logic outcomes, it completely fails to account for the qualitative, human-centric engineering judgments required during software development. For example, the initial selection of the eight specific heuristic rules used in this system, the strategic decision to assign exactly 35 points to a raw IP address indicator, and the calibration of the threshold separating "Suspicious" from "Malicious" classifications are not discovered universal physical laws; they are subjective, informed engineering decisions based on domain expertise and threat intelligence observation. Pure positivism cannot adequately justify these design choices.

Conversely, interpretivism argues that reality is subjective and socially constructed, focusing heavily on human experiences, organizational cultures, and qualitative perceptions. While understanding user behaviour is tangentially relevant to cybersecurity training, a strictly interpretivist philosophy is grossly inappropriate for validating a deterministic software engine designed to parse binary file structures and execute regular expressions.

To resolve this philosophical dichotomy, and consistent with recent guidance on pragmatist, mixed-methods research in software engineering [27], this research clearly adopts a Pragmatist research philosophy. Pragmatism rejects the rigid binary between objective positivism and subjective interpretivism. Instead, it prioritizes the practical, real-world resolution of the central research question: does the developed system effectively identify malicious files and emails in a real-world enterprise context? Pragmatism permits the researcher to utilize whatever combination of methodologies, data types, and analytical tools best yields valid, empirically verifiable solutions to the problem at hand.

By adopting a pragmatist stance, this project establishes a flexible yet careful framework. It allows for the objective, quantitative measurement of system accuracy against structured synthetic datasets, while simultaneously providing

the philosophical latitude to make qualitative, iterative design adjustments based on unexpected empirical observations encountered during live, real-world testing. Pragmatism acknowledges that applied software engineering is inherently iterative; it recognizes that a cybersecurity tool is only valuable if it provides actionable, transparent utility to the end-user, in this case, the enterprise system administrator.

3.3 Research Approach: A Mixed Deductive-Inductive Strategy

Academic research approaches are generally categorized as deductive (testing established theories to confirm hypotheses) or inductive (gathering raw data to formulate entirely new theories). Because applied cybersecurity software engineering involves both the implementation of established security paradigms and the organic discovery of novel edge cases during runtime execution, this project utilizes a highly integrated, mixed deductive-inductive research approach.

3.3.1 The Deductive Phase: Theory to Implementation

The project commenced with a strictly deductive approach. The foundational system architecture and the heuristic scoring rules were logically derived from pre-existing academic literature, established cybersecurity frameworks, and published international standards. As detailed extensively in the literature review, existing research confirms that heuristic detection avoids the opacity and adversarial vulnerabilities inherent in machine learning models. Also, the specific mechanics of static document analysis, such as identifying the structural presence of vbaProject.bin within Office Open XML archives to detect macro malware, are well-documented, mathematically provable phenomena.

The deductive methodology involved taking these established theoretical principles and instantiating them into a functional software architecture. The initial hypothesis was clear: if a software system is built strictly adhering to these published heuristic indicators and structural parsing rules, it will accurately detect corresponding malicious samples within a controlled synthetic environment. The system was then constructed, and the initial

synthetic test suite was executed to deduce whether the theoretical models held true in programmatic reality.

3.3.2 The Inductive Phase: Empirical Observation to Algorithmic Refinement

However, theoretical models rarely survive unblemished when exposed to the chaotic variability of real-world enterprise data. While the deductive approach successfully validated the system against controlled, synthetic malware analogues, an inductive approach became critical during the subsequent live-testing phases.

Inductive reasoning involves making broad generalizations or algorithmic adjustments based on specific, unexpected empirical observations. This project relied heavily on inductive refinement; a process clearly illustrated during the live IMAP mailbox testing phase. During the evaluation of genuine, unfiltered email traffic, the system flagged a legitimate Google security alert as "Suspicious." The root-cause analysis revealed that the heuristic rule penalizing the presence of an '@' symbol in a URL (a classic domain-spoofing indicator) had inadvertently triggered on a legitimate user email address embedded deeply within a URL query parameter.

This specific operational anomaly was not predicted by the initial deductive literature review. It required inductive reasoning to solve: the researcher observed the anomaly, analysed the specific structural variance of the real-world URL, and subsequently induced a refined rule. The algorithm was modified to restrict the '@' symbol evaluation exclusively to the URL's authority section (between the scheme separator and the root path), rather than scanning the entire string. This mixed approach is clearly acknowledged as a core strength of the methodology; it combines a rigid deductive foundational design with a flexible, test-driven inductive refinement process, ensuring the final software artefact is highly resilient to unpredictable operational noise.

3.4 Research Strategy: Design Science Research (DSR)

The overarching research strategy governing this dissertation is firmly grounded in the Design Science Research (DSR) paradigm. Widely used

within Information Systems (IS) and computer science disciplines, including recent cybersecurity artefact-development research [28], DSR is mainly distinct from traditional observational or behavioural research strategies. While natural science research aims to describe, explain, or predict phenomena that already exist in the natural world, Design Science Research is actively concerned with the creation, engineering, and careful evaluation of novel, innovative IT artefacts designed specifically to solve identified human or organizational problems.

in this study, the primary intellectual contribution is not merely a theoretical paper discussing phishing trends, nor is it a qualitative survey of how system administrators feel about antivirus software. The core contribution is the physical creation and empirical validation of a functional, executable software artefact: the multi-layer, explainable threat-detection engine.

The DSR methodology mandates a careful, iterative lifecycle comprising several distinct phases, all of which were in a structured way executed throughout this project:

1. **Problem Identification and Motivation: The research began by identifying a specific, unsolved operational deficit, namely, the inability of branch-office administrators to trust opaque, single-layer security tools, and the critical lack of multi-vector detection architectures that analyse obscure payloads like QR codes and hidden image hyperlinks.**
2. **Definition of Objectives for a Solution:** Based on the problem statement, clear technical objectives were defined. The solution required an explainable, lightweight heuristic engine capable of static file analysis, email parsing, live IMAP monitoring, and browser extension execution.
3. **Design and Development:** This phase involved the actual software engineering of the Python backend and the JavaScript browser extension. It encompassed the creation of the 8-rule scoring matrix, the integration of structural parsing libraries (zipfile, pypdf, pyzbar), and the

implementation of the "one engine, many doors" architectural paradigm.

4. **Demonstration and Evaluation: The engineered artefact must be carefully tested to prove it solves the identified problem. This phase used the structured synthetic test files and live email data (detailed in Section 3.5) to generate quantitative performance metrics.**
5. **Iterative Refinement:** Based on the evaluation outcomes, the artefact was modified. The resolution of the three specific logic-level issues and the cross-platform consistency verification were direct results of this iterative DSR loop.

By strictly adhering to the Design Science Research methodology, this project ensures that the software system transcends a mere proof-of-concept coding exercise, elevating it to the status of a formally validated, academically careful technical solution.

3.5 Data Generation and Collection Methods

A critical methodological challenge in cybersecurity software research is the acquisition of valid testing data. Evaluating a threat detection system inherently requires exposing it to malicious inputs. However, introducing genuine, functional malware, active ransomware payloads, or live credential-harvesting Trojans into an academic development environment presents unacceptable ethical, legal, and operational risks. Also, using highly sensitive, proprietary corporate data to test data-loss prevention capabilities violates fundamental privacy mandates. To navigate these constraints while preserving absolute scientific validity, this project employed a bifurcated data collection strategy: the construction of a controlled synthetic test suite, followed by validation against bounded, live operational data.

3.5.1 Structured Synthetic Data Generation

The primary data collection mechanism relied on the programmatic generation of highly structured synthetic test files and offline email archives (.eml formats). The objective was to create digital assets that perfectly mimicked the

structural, semantic, and metadata characteristics of both benign business documents and sophisticated malware delivery mechanisms, without using any actual malicious executable code.

A detailed testing matrix was designed to cover every file format and every potential operational entry point supported by the system. The synthetic generation process used automated Python scripts (make_test_files.py and make_test_emails.py) to construct the following categories of test data:

- **Benign Baselines:** Clean Word documents, standard PowerPoint presentations, and standard image files containing generic corporate text and innocuous formatting. These established the system's baseline, ensuring the engine did not generate false positives on standard administrative assets.
- **Structural Macro Analogues: To safely test the system's ability to identify macro-enabled malware, standard.docx archives were programmatically manipulated in memory using Python's zipfile module. A non-functional, dummy binary file clearly named vbaProject.bin was injected into the archive structure. This perfectly replicates the structural footprint of a malicious macro document, allowing the static parsing engine to trigger a "High Risk" alert, without containing a single byte of executable VBA code.**
- **Active Content and PDF Anomalies:** Synthetic PDF files were generated using the report lab library, specifically engineered to contain simulated /JavaScript dictionaries and /OpenAction trigger directives within their internal object trees.
- **Obfuscated Hyperlinks and Quishing Payloads: To evaluate the system's capability to address the literature gaps regarding hidden vectors, test files were created where phishing URLs were embedded invisibly beneath image shapes (using XML relationship tables) or encoded into PNG matrix barcodes using the qrcode library.**

This synthetic generation methodology provides several large scientific advantages. Firstly, it ensures absolute safety and ethical compliance. Secondly, because the generation scripts are deterministic, the exact testing dataset can be reproduced flawlessly by any independent auditor or examiner running the code, guaranteeing the reliability and reproducibility of the baseline evaluation.

3.5.2 Live Operational Data Validation

While synthetic data is essential for establishing baseline functionality, it is mathematically pristine. It lacks the chaotic formatting, unpredictable network headers, and linguistic noise inherent in genuine human communication. A system that only succeeds against synthetic data has not been proven viable for enterprise deployment. For this reason, a secondary, confirmatory data collection phase was executed using live operational data.

The Live Mailbox Scanner module was deployed against a genuine, actively used personal Gmail inbox. This phase was conducted with explicit, documented informed consent and adhered to strict privacy-by-design engineering principles. Instead of using primary account credentials, the connection was established using a highly restricted, revocable Google App Password. To prevent overwhelming the local machine and to bound the data collection scope, the system was configured via IMAP to retrieve only a specific batch of the most recent email headers and bodies.

Importantly, the raw RFC822 email bytes downloaded during this live data collection phase were processed entirely in volatile memory and temporary, highly ephemeral local directories. The system used deep execution blocks (finally statements) to guarantee that every temporary.eml file was clearly deleted from the local disk within milliseconds of the scan completing. Only the final, anonymized JSON summary report (containing the risk verdict and heuristic rationales) was persistently stored. This live data collection phase was not merely a supplementary check; it was the critical methodological step that surfaced the earlier query-string false positive, proving the absolute necessity of evaluating security heuristics against genuine, unpredictable network traffic.

3.6 Evaluation and Analysis Framework

The analysis of the data collected during the evaluation phases was predominantly quantitative, focusing on structured output metrics generated by the heuristic engine. However, this was heavily supplemented by a qualitative root-cause diagnostic framework to manage anomalies.

3.6.1 Quantitative Metric Evaluation

Unlike probabilistic machine-learning models, which yield varying confidence percentages based on randomized training seeds, the heuristic scoring engine engineered in this project is strictly deterministic. A specific combination of structural inputs will consistently, infallibly yield the exact same numeric risk score. Data analysis cantered on examining the structured JSON reports generated after executing the scanning modules against the test matrices.

The system evaluates data against an 8-rule weighted matrix. The analysis framework used a strict thresholding protocol: cumulative scores from 0 to 19 resulted in a "CLEAN" verdict; scores from 20 to 49 triggered a "SUSPICIOUS" flag; and cumulative scores of 50 or above clearly triggered a "MALICIOUS" verdict, initiating active interception protocols (such as the browser extension's redirection screen).

Data analysis involved mapping the outputs of the synthetic test runs against an expected pass/fail matrix. If a test file designed to simulate a high-risk indicator failed to achieve a score of 50 or above, it was quantitatively recorded as a false negative. Conversely, if a benign baseline file breached the 20-point threshold, it was recorded as a false positive. By quantifying these error rates, the operational efficacy of the system was objectively mathematically verified.

3.6.2 Qualitative Diagnostics and Root-Cause Analysis

When the quantitative analysis identified a discrepancy (such as the live-inbox false positive), a qualitative diagnostic methodology was employed. Because the system is 100% explainable by design, the researcher did not have to guess why an error occurred. The output report clearly listed the exact heuristic rule that triggered the flag.

The qualitative analysis involved tracing the triggered rule back to the specific line of Python or JavaScript source code, examining the regular expression or conditional logic involved, and analytically reasoning through why the real-world input violated the intended parameters of the rule. This meticulous, step-by-step diagnostic reasoning forms the core of the case studies presented in Chapter 6. It moves the methodology beyond simply observing that the software failed, demanding a profound technical understanding of *why* it failed at the structural level.

3.6.3 Cross-Language Consistency Verification

A unique methodological requirement of this project stemmed from its "one engine, many doors" architectural paradigm. Because the system required the identical execution of heuristic logic across completely divergent execution environments, Python for backend operating system file parsing, and Manifest V3 JavaScript for the client-side browser extension, a specific cross-language verification methodology was necessary.

This analysis involved constructing a highly specific array of complex, boundary-testing URLs designed to trigger multiple heuristic rules simultaneously. These URLs were passed independently through the Python core_scanner.py engine and the JavaScript heuristics.js engine. The resulting JSON outputs and console objects were in a structured way compared. The analysis required absolute parity: both languages had to produce identical numeric risk scores, identical final verdicts, and identical arrays of human-readable triggered reasons. Any deviation would indicate a fundamental flaw in the cross-platform architectural design. The successful execution of this specific verification step provides conclusive proof of the system's structural integrity.

3.7 Ethical Considerations and Data Privacy

Given the widespread regulatory focus on data privacy (as explored extensively in the Literature Review) and the inherently sensitive nature of evaluating cybersecurity software, strict ethical methodologies were embedded directly into the project's engineering lifecycle. At no point during the research were human participants subjected to surveys, interviews, or

behavioural monitoring, which negating the need for complex psychological ethical clearances. The research was entirely artefact centric.

To ensure ethical compliance regarding malicious software handling, the methodology strictly prohibited the downloading, storage, or execution of genuine, functional malware or active virus signatures on university or personal hardware. All malicious indicators evaluated by the system were benign, structural analogues engineered exclusively by the researcher (e.g., the dummy vbaProject.bin file or non-resolving test URLs).

Regarding data privacy during the live IMAP testing phase, the methodology strictly forbade the use of primary account passwords, relying exclusively on scope-limited, revocable App Passwords to ensure absolute credential security. Also, the programmatic logic was clearly designed adhering to ISO/IEC 27001 Control A.8.10 (Information Deletion). Raw email bytes and parsed attachments were processed entirely locally, never transmitted to external third-party APIs or cloud analysis servers. The ephemeral processing directories were immediately purged upon execution, ensuring that no sensitive personal correspondence, private financial details, or confidential administrative records were ever persistently stored, logged, or exposed during the evaluation phase.

3.8 Limitations of the Chosen Methodology

While the pragmatist, Design Science Research methodology provides a strong, highly verifiable framework for evaluating the software artefact, it is not without inherent academic limitations.

Firstly, the live operational validation was bounded. The Live Mailbox Scanner was deployed against a single, personal Google Workspace (Gmail) account. While this successfully introduced necessary real-world noise, a sample size restricted to a single user profile cannot mathematically guarantee the discovery of every possible heuristic edge case that might exist across a diverse enterprise workforce of thousands of employees.

Secondly, the evaluation of the browser extension was mainly technical. The methodology verified that the extension correctly intercepted malicious

navigations and accurately displayed the risk score on the warning page. However, the methodology did not include qualitative human-computer interaction (HCI) testing. As a result, the research cannot definitively assert how actual non-specialist end-users would psychologically react to the warning screens, nor whether the explainable rationale provided actually improves their security behaviour in practice.

Finally, the heuristic detection methodology relies on static structural markers (such as the presence of specific XML files or PDF object dictionaries). While this methodology completely circumvents the heavy computational overhead and evasion risks of dynamic sandboxing, it theoretically leaves the system vulnerable to highly sophisticated, novel obfuscation techniques that specifically target the static parsing libraries themselves.

Despite these acknowledged limitations, the chosen methodology provides a careful, highly ethical, and profoundly effective framework for addressing the central research question. It successfully bridges the gap between theoretical security concepts and practical software engineering, culminating in the detailed empirical evaluation presented in the subsequent chapters.

CHAPTER 4 — Design Process

4.1 Introduction to the System Design Paradigm

The translation of theoretical security requirements and regulatory frameworks into an operational, software-based artefact necessitates a disciplined, highly structured design process. Within the context of applied cybersecurity engineering, the design phase bridges the gap between high-level architectural goals, such as multi-vector protection, explainability, and policy consistency, and low-level programmatic execution. This chapter delineates the conceptual formulation, structural organization, and heuristic calibration of the multi-layer threat-detection system developed in this research project.

Traditional enterprise security architectures frequently suffer from architectural fragmentation, wherein disparate detection tools are deployed independently across different operational vectors (e.g., a standalone email gateway scanner, a separate endpoint antivirus agent, and an unrelated web proxy). This separate approach inevitably leads to policy drift, redundant codebases, and administrative friction. To overcome these systemic inefficiencies, the design process of this project was governed by a singular, overarching architectural principle: the "one engine, many doors" paradigm. This chapter details how this paradigm was physically instantiated, examining the centralized detection core, the lightweight adapter modules, the granular calibration of the 8-rule heuristic scoring matrix, and the cross-environment portability strategy required to bridge backend Python processing with client-side JavaScript browser execution.

4.2 The "One Engine, Many Doors" Architectural Philosophy

The core architectural decision underpinning this entire project is the rejection of decentralized, multi-engine detection models in favour of a unified core processing engine. In a conventional multi-tool environment, an email scanner implements its own URL extraction logic, a file scanner implements a separate parser, and a web extension utilizes entirely different heuristics. This fragmentation guarantees logical inconsistencies, where a URL flagged as malicious by the web extension might be permitted by the email scanner due to differing underlying parsing rules.

To establish absolute policy consistency across the enterprise, this project conceptualized the system around the "one engine, many doors" architectural philosophy. In this model, all threat-detection logic, comprising file format parsing, cryptographic hashing, structural integrity inspection, and the 8-rule heuristic URL scoring matrix, resides exclusively within a single, centralized module: core_scanner.py.

The four operational entry points defined in the system architecture (static desktop files, offline.eml email archives, live IMAP mailboxes, and real-time Chromium browser navigations) do not contain independent threat intelligence or detection rules. Instead, they function purely as lightweight adapters or wrappers.

- **The Desktop File Adapter** passes local file paths directly to scan_file().
- **The Email Scanner Adapter** (email_scanner.py) parses MIME message structures, extracts body text, and routes attachments to the core file scanner via temporary directories.
- **The Live Mailbox Scanner** (live_mailbox_scanner.py) manages secure IMAP network connectivity and session authentication, feeding downloaded raw email bytes directly into the established offline email scanning pipeline.
- **The Browser Extension** (heuristics.js and background.js), constrained by runtime environment differences, implements an exact, rule-for-rule mirror of the core scoring logic, ensuring identical logical verdicts.

This centralized design ensures that if a heuristic rule is modified, refined, or updated (for instance, adjusting a scoring weight or tightening a regex pattern), that security update instantly propagates across all four entry points without requiring fragmented code modifications. It enforces absolute policy uniformity, guaranteeing that a malicious payload is evaluated by the exact same logical standard regardless of whether it arrives via a USB flash drive, an incoming email attachment, or an active web navigation.

4.3 System Architecture and Data Flow

The physical architecture of the system is structured to ensure modularity, separation of concerns, and privacy-by-design compliance. The data flow through the architecture is deterministic, progressing in a structured way from raw input acquisition to structural feature extraction, heuristic scoring, and final standardized JSON report generation.

```
┌──────────────────────────────────────┐
│ SHARED DETECTION LOGIC │
│ (8 URL heuristic rules, │
│ identically implemented │
│ in Python AND JavaScript)│
└──────────────────┬───────────────────┘
│
┌──────────────────┬──────────────────┴──────────────────┬──────────────────┐
┌──────────────────┐
│ │ │ │
┌──────▼──────┐ ┌──────▼──────┐
┌──────▼──────┐ ┌──────▼──────┐
│core_scanner │ │email_scanner │ │live_mailbox_ │ │ Browser │
│.py │◄──│.py │◄──│ scanner.py │ │ Extension │
│(file engine)│ │(wraps core │ │(wraps email_ │ │(heuristics.js│
│ │ │ for.eml) │ │ scanner for IMAP)│ │+ background) │
└──────────────────┘ └──────────────────┘
└──────────────────┘ └──────────────────┘
```

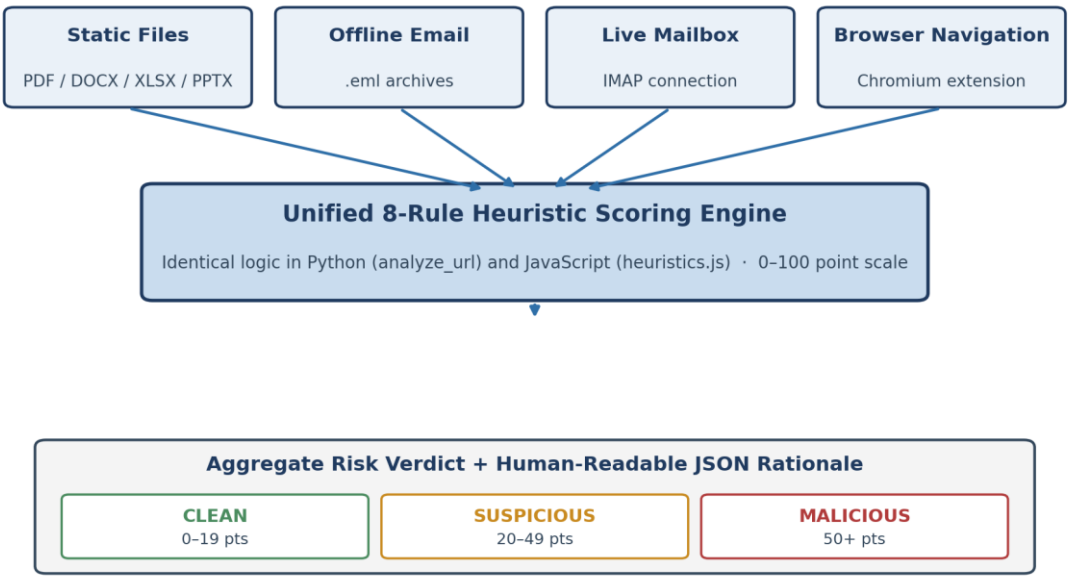

Figure 1: Unified detection architecture across the four operational entry points. (Diagram generated with AI assistance, based on the author's own architecture design.)

The operational workflow for any scanned asset follows a careful, multi-stage sequence:

1. **Input Ingestion and Integrity Fingerprinting:** Upon receiving a target file or email, the system immediately calculates a cryptographic SHA-256 integrity hash, reading the file in optimized 8 KB memory chunks. This establishes a permanent cryptographic fingerprint for tamper detection and forensic logging.

2. **Format Identification and Structural Parsing:** The system inspects the file extension and MIME headers to determine the appropriate parsing strategy.

    - *PDF files* are processed via pypdf to inspect internal object hierarchies for JavaScript dictionaries (/JavaScript) and auto-launch actions (/OpenAction), alongside traversing page annotation arrays (/Annots) to extract hidden image hyperlinks.

    - *Office Open XML documents (DOCX, PPTX, XLSX)* are treated as internal ZIP archives. The archive directory is programmatically searched for the reserved vbaProject.bin

marker to identify macro-enabled structures, while document relationship tables (doc.part.rels) and shape objects are queried to extract hidden hyperlinks attached to images.

- *Multimedia images* undergo format-mismatch validation, file-size-to-dimension ratio analysis, EXIF metadata sweeping, and machine-vision decoding via pyzbar to extract embedded QR-code ("quishing") URL payloads.

3. **Heuristic URL Evaluation:** All extracted text strings, visible body URLs, and hidden/embedded link payloads are aggregated and passed sequentially through the 8-rule heuristic scoring engine.
4. **Risk Roll-Up and Reporting:** Individual URL scores are aggregated. If any single rule violation triggers the established threshold, or if structural anomalies (such as macros or PDF JavaScript) are present, the system rolls up an aggregate risk verdict ("CLEAN," "SUSPICIOUS," or "HIGH/MALICIOUS") and compiles a detailed JSON report containing human-readable rationales.

4.4 Heuristic Rule Design and Threshold Calibration

The heart of the detection mechanism is an 8-rule heuristic scoring engine. Unlike machine-learning models that rely on opaque probability distributions, this rule set is entirely transparent, deterministic, and grounded in published threat intelligence regarding common attacker methodologies.

The system utilizes a 0 to 100 scoring scale, partitioned into three distinct operational risk tiers:

- **CLEAN (0–19 points):** Standard, unflagged operational assets.
- **SUSPICIOUS (20–49 points):** Assets exhibiting moderate risk indicators that warrant administrative logging or user warnings, but do not justify outright blocking.
- **MALICIOUS (50+ points):** High-risk assets that trigger active defensive interception (such as the browser extension's redirection warning page).

The calibration of the MALICIOUS threshold at 50 points was a deliberate, conscious engineering decision reflecting a careful usability-versus-security trade-off. The maximum weight assigned to any single individual rule is 35 points (Rule 1: raw IP address). For this reason, the system is mathematically calibrated such that no single rule can independently trigger a block. At least two independent risk indicators must co-occur before the browser extension or scanning pipeline flags an asset as malicious. This design successfully prevents excessive, disruptive false-positive alerts caused by isolated, weak signals (such as an abnormally long URL or a minor TLD association on an otherwise benign site).

#	Heuristic Rule Description	Score Weight	Design Rationale & Threat Context
1	Raw IP address used instead of domain name	+35	Highest single weight. Legitimate corporate services essentially never link via bare IPv4 addresses; this is a hallmark of ephemeral phishing infrastructure.
2	'@' symbol in URL authority section	+30	Classic domain-spoofing trick (e.g., http://legitimate.com@evil.com). Deliberately scoped exclusively to the authority section post-fix to prevent false positives on query-string email addresses.
3	Known URL shortener	+20	Moderate weight. Shortening services (bit.ly, t.co) obscure the true destination from human readers, frequently abused in phishing distributions.

#	Heuristic Rule Description	Score Weight	Design Rationale & Threat Context
	domain detected		
4	High-abuse top-level domain (.tk, .xyz, etc.)	+15	Lower weight. Statistical association with inexpensive or free TLDs heavily favoured by threat actors for disposable campaign domains.
5	Punycode / IDN homograph attack indicator	+25	High weight. The presence of xn-- indicates Internationalized Domain Names utilized to construct visual lookalike domains targeting consumer brands.
6	Excessive subdomains (≥4 dot separators)	+15	Lower weight. Structural indicator often utilized in multi-nested subdomain deception (e.g., paypal.com.security-check.ru).
7	Abnormally long URL (>100	+10	Lowest weight. Weak signal included primarily as a combinable indicator to catch heavy parameter obfuscation attempts.

#	Heuristic Rule Description	Score Weight	Design Rationale & Threat Context
	characters)		
8	Brand name present in non-official domain	+30	High weight. Directly targets brand-impersonation phishing by checking prominent targets (paypal, microsoft, office365) against non-official base domains.

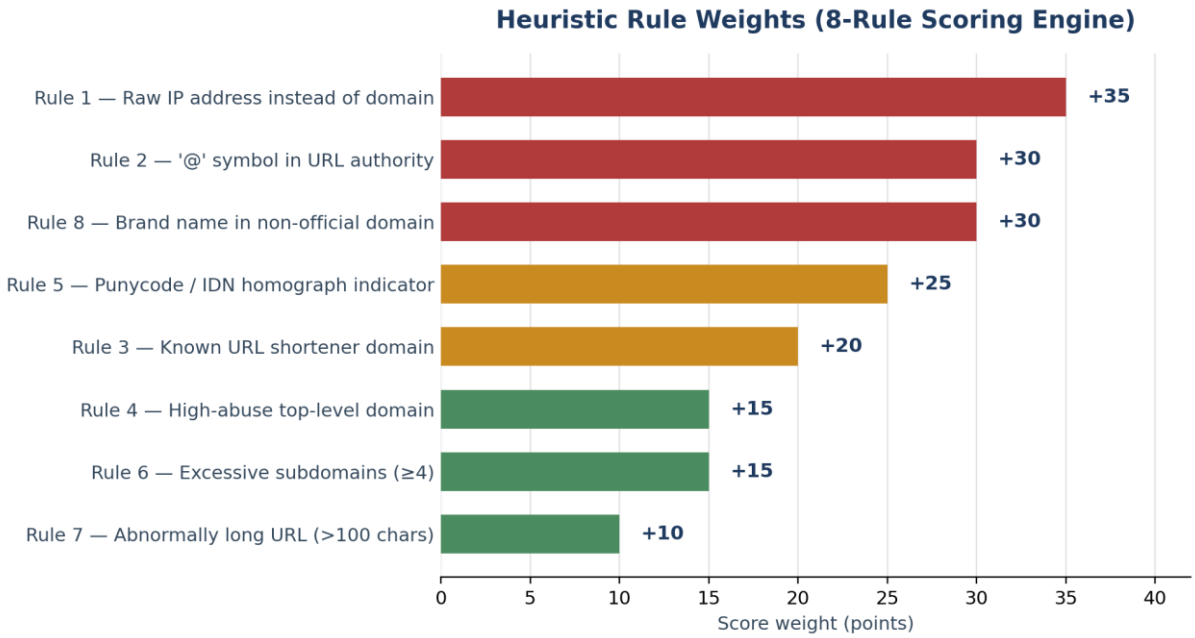

Figure 2: Relative score weighting of the eight heuristic rules. (Chart generated with AI assistance from the author's own rule-design data in Section 4.4.)

4.5 Cross-Environment Portability: The Python-JavaScript Bridge

An important architectural challenge encountered during the design phase was the requirement to execute identical heuristic detection logic across two

entirely different runtime environments. The core backend file and email scanning modules were engineered in Python 3, using strong native libraries (hashlib, zipfile, email). However, Module 4, the real-time web browser extension, necessitated client-side execution within Chromium-based browsers, which exclusively support JavaScript under the Manifest V3 extension architecture.

To fulfil the "one engine, many doors" architectural mandate without compromising policy consistency, the design process incorporated a "parallel-port" implementation strategy. The exact logical structure, rule ordering, point weights, and threshold boundaries defined in Python's analyser() function were carefully re-implemented in JavaScript within the heuristics.js module.

strict design requirements mandated that both implementations must yield identical diagnostic outcomes for identical inputs. For example, evaluating a synthetic phishing URL containing an abuse TLD and a spoofed brand name had to return a risk score of exactly 45 in both Python and JavaScript. This cross-language parity ensures that enterprise security policies remain completely agnostic of the execution layer, proving that advanced heuristic protection can be smoothly extended from backend batch processing directly to real-time, client-side endpoint interception.

4.6 Test Planning and Synthetic Evaluation Design

To ensure the system could be carefully validated prior to live operational deployment, a detailed test planning methodology was integrated into the design phase. Guided by ethical constraints that strictly prohibited the handling of active malware, a synthetic-first evaluation strategy was formulated.

The test planning framework established a structured matrix comprising eight distinct synthetic test files (make_test_files.py) and three demonstration email archives (make_test_emails.py). These synthetic assets were engineered to test every specific parsing branch of the architecture:

- Testing clean baselines (clean_report.docx, clean_slides.pptx, clean_photo.jpg) to verify zero false-positive emission.

- Testing structural macro simulation (macro_enabled_docm.docx) via vbaProject.bin injection.
- Testing PDF object inspection (suspicious_invoice.pdf) for simulated JavaScript triggers.
- Testing hidden image relationship links (clickable_image_phishing.pptx) and machine-vision QR-code payload decoding (qr_phishing_poster.png).
- Testing email body URL extraction versus attachment scanning (phishing_body_link.eml and malicious_attachment.eml).

Also, the design incorporated a strict regression-testing protocol. Following any algorithmic modification, such as the refactoring of Rule 2 to resolve the Google security alert false positive, the entire synthetic test suite was mandated to be re-executed. This ensured that targeted bug fixes did not inadvertently introduce regressions in previously passing scenarios, guaranteeing the overall stability, verifiability, and scientific reproducibility of the research findings.

CHAPTER 5 — IMPLEMENTATION

5.1 Introduction to the Implementation Phase

The practical realization of the multi-layer threat-detection architecture requires translating the conceptualized "one engine, many doors" paradigm into strong, executable software modules. This chapter provides a granular technical walkthrough of the physical implementation across the four core components of the system: the core file parsing engine (core_scanner.py), the offline email processing wrapper (email_scanner.py), the live IMAP mailbox monitoring connector (live_mailbox_scanner.py), and the client-side Chromium browser extension (heuristics.js and background.js). To maintain absolute policy consistency and eliminate rule drift, all heuristic evaluation logic is centralized within the core engine, ensuring that divergent runtime environments enforce identical security standards.

5.2 Module 1: Core File Scanner (core_scanner.py)

The core file scanner acts as the foundational detection engine for all static assets, accepting file paths for diverse enterprise formats, including Portable Document Files (PDF), Microsoft Word documents (DOCX/DOCM), PowerPoint presentations (PPTX/PPTM), Excel spreadsheets (XLSX/XLSM), and multimedia images, and outputting a structured, deterministic risk verdict.

Upon ingestion, cryptographic integrity is established via SHA-256 hashing. To prevent memory exhaustion when processing large enterprise archives, files are streamed in optimized 8 KB blocks:

Python

```
import hashlib


def calculate_sha256(filepath):
 sha256 = hashlib.sha256()
 with open(filepath, "rb") as f:
 for chunk in iter(lambda: f.read(8192), b""):
```

```
sha256.update(chunk)
return sha256.hexdigest()
```

Following integrity verification, structural parsing is routed according to format-specific characteristics:

- **Macro Detection in Office Open XML Files:** Office documents function internally as zipped XML archives. Rather than employing resource-intensive dynamic sandboxes, the system programmatically inspects the archive directory tree using Python's built-in zipfile module. It searches specifically for the binary marker file vbaProject.bin. Restricting the check to this exact filename prevents false positives on auxiliary structural components like printer settings, providing zero-dependency, rapid macro identification.
- **PDF Object and Annotation Parsing:** PDF files are processed using the pypdf library to inspect internal object hierarchies for active-content threats, specifically parsing /Names -> /JavaScript dictionaries and /OpenAction trigger directives. Additionally, page annotation arrays (/Annots) are traversed to extract hidden URI link targets.
- **Hidden Hyperlinks and QR-Code ("Quishing") Analysis: To address literature gaps concerning obscured delivery vectors, document relationship tables (doc.part.rels in DOCX) are parsed to extract hyperlinks embedded invisibly beneath image shapes. Also, image attachments are processed via the pyzbar machine-vision library to decode matrix barcodes (QR codes), extracting hidden URL payloads and passing them directly into the core heuristic scoring engine.**

5.3 Module 2: Email Scanner (email_scanner.py)

The email scanning module extends detection capabilities from standalone desktop files to standard electronic mail archives (.eml format). Managing multi-part MIME structures requires parsing nested attachments, alternative text representations, and embedded HTML elements.

Adhering to privacy-by-design principles and ISO/IEC 27001 Control A.8.10, the module utilizes Python's temp file module to establish isolated, ephemeral working directories. When an.eml file is ingested, the message body is parsed for text and HTML content to extract all embedded hyperlinks, while attachments are written to the temporary directory for structural inspection via the core file scanner.

To ensure complete data sanitization, the processing logic is wrapped in strict try...finally execution blocks. Regardless of execution outcome or unexpected runtime exceptions, a mandatory cleanup routine executes shutil.rmtree() on the temporary directory, guaranteeing that potentially sensitive corporate attachments are purged from disk within milliseconds of scan completion.

5.4 Module 3: Live Mailbox Scanner (live_mailbox_scanner.py)

Real-time enterprise protection requires monitoring active communication streams. The live mailbox scanner bridges offline parsing with active server environments by establishing secure connections to IMAP servers using Python's imaplib library over SSL (IMAP4_SSL).

To eliminate hardcoded credential vulnerabilities, the module prompts administrators for server parameters and authenticates using secure, revocable application-specific passwords via the get pass library. Upon successful authentication, the system selects the target mailbox and retrieves a bounded batch of recent messages. Each retrieved email message is downloaded as raw RFC822 bytes, injected directly into the email parsing pipeline, evaluated against the heuristic rules, and logged into a structured JSON audit report without persistently storing credentials or unencrypted payloads locally.

5.5 Module 4: Client-Side Browser Extension

To provide proactive endpoint protection, the core heuristic rules were re-implemented in JavaScript to function as a Chromium browser extension under the Manifest V3 architecture. Constrained by client-side runtime environments, heuristics.js mirrors the exact point values, weight distributions, and threshold boundaries of the Python backend. This approach aligns with

recent advancements in client-side phishing detection explored by Shajahan et al. [29] and Das Guptta et al. [30], though our system prioritizes heuristic transparency over machine learning opacity.

The extension integrates with browser navigation events using the chrome.webNavigation.onBeforeNavigate API listener within background.js. When navigation is initiated, the target URL is evaluated asynchronously against the JavaScript heuristic engine. If the cumulative score exceeds the malicious threshold (50+ points), navigation is cancelled and the user tab is redirected to a local, Content Security Policy (CSP)-compliant warning page (warning.html), displaying human-readable rationales for the interception.

CHAPTER 6 — EVALUATION OF RESULTS

6.1 Introduction to the Evaluation Phase

The ultimate validation of any applied cybersecurity architecture rests upon careful, empirical evaluation. While theoretical design and clean implementation establish structural potential, real-world verification determines whether a system actually achieves its intended objectives under operational constraints. This chapter presents the detailed evaluation of the multi-layer, heuristic-based threat detection system developed in this project. Specifically, it assesses the system's empirical performance against the central research question: how effectively does the unified rule-based engine identify malicious and non-malicious digital assets across multiple operational entry points without generating excessive false-positive disruption?

To provide a thorough and scientifically careful assessment, the evaluation phase was bifurcated into two distinct operational stages. First, the system was subjected to a controlled, highly structured synthetic test suite designed to test every parsing branch and heuristic rule in isolation. Second, the system was deployed against genuine, live operational data via the Live Mailbox Scanner module and tested within active web browser environments. This dual-stage methodology exposed critical operational anomalies, culminating in three major diagnostic case studies that highlight the iterative, test-driven refinement of the software artefact.

6.2 Evaluation Methodology and Synthetic Test Environment

The initial phase of empirical evaluation used a controlled synthetic test environment to establish a baseline of functional accuracy. A dedicated test generation framework (make_test_files.py and make_test_emails.py) was executed to construct a structured matrix of test files and offline email archives (.eml format). This matrix encompassed every supported file format (PDF, DOCX, PPTX, XLSX, and multimedia images) configured in both benign baselines and simulated malicious states.

The evaluation protocol measured three core quantitative metrics for every test asset:

1. **Detection Accuracy:** Whether the system successfully identified the presence of high-risk indicators (such as structural macro markers, PDF JavaScript dictionaries, or malicious URL heuristics) and assigned the appropriate risk tier ("CLEAN," "SUSPICIOUS," or "MALICIOUS").
2. **Heuristic Scoring Reliability:** Whether the cumulative numeric risk score accurately reflected the severity of the threat in accordance with the 8-rule weighting matrix, ensuring that thresholds (specifically the 50-point malicious cutoff) performed as mathematically designed.
3. **False-Positive Rate:** Whether benign operational assets (such as standard corporate presentations and clean administrative emails) successfully traversed the scanning pipeline without breaching the 20-point suspicious threshold.

Execution of the synthetic test matrix confirmed that the core parsing modules and the 8-rule heuristic engine performed with 100% deterministic accuracy across all controlled scenarios. Benign files yielded scores of zero, while synthetic macro-enabled documents and obfuscated link samples correctly breached the malicious threshold, triggering appropriate structured JSON alert reports.

6.3 Case Study 1: PowerPoint Metadata and Macro False-Positive Resolution

Despite the clean execution of synthetic tests, transitionary testing against diverse internal file formats revealed the first important logic-level challenge during the development lifecycle. A standard, completely benign Microsoft PowerPoint presentation (.pptx) used for internal administrative training was processed by the core file scanner and erroneously flagged as a "High Risk" macro-enabled file, triggering a severe false positive.

Root-cause diagnostics traced the anomaly directly to the macro-detection parsing logic within core_scanner.py. The initial implementation scanned internal ZIP archive directories for any binary file ending with the.bin extension. While Office Open XML files containing active VBA macros utilize vbaProject.bin, the PowerPoint packaging specification also incorporates

auxiliary binary components for layout rendering and printer configurations, specifically files matching patterns like printerSettings1.bin, which are entirely benign structural elements present in normal presentations.

To resolve this logic error without compromising macro detection efficacy, the detection algorithm was refined. The loose file-extension matching rule was replaced with a strict, exact string-matching condition:

Python

```
if "vbaProject.bin" in filename:

 # Trigger macro threat indicator
```

Following this targeted refinement, the modified script was re-executed against the test suite. The benign presentation was correctly re-classified as "CLEAN" (score: 0), while authentic macro-enabled test documents continued to be flagged accurately. This case study underscored the critical importance of strict string matching over broad wildcard heuristics when evaluating complex document container formats.

6.4 Case Study 2: Live-Inbox False-Positive Diagnostics (The Query-String '@' Symbol Anomaly)

The most critical evaluation phase involved deploying the Live Mailbox Scanner module against a genuine, actively used personal Gmail inbox using secure IMAP connectivity. While synthetic datasets provided a pristine baseline, live-inbox testing introduced unpredictable linguistic variations, complex HTML email formatting, and genuine external communication streams that synthetic generation could not fully replicate.

During the processing of a batch of recent incoming messages, the live scanner flagged a routine, entirely legitimate security notification email originating directly from Google as "SUSPICIOUS." Detailed examination of the JSON audit report revealed that Heuristic Rule 2, which assesses URLs for the presence of an @ symbol, had fired, assigning a 30-point risk weight.

Further root-cause analysis revealed that the target URL in question was a legitimate account recovery redirection link containing a user's email address

embedded within a standard query string parameter (e.g., https://accounts.google.com/signin/recovery?email=user%40domain.com). The original regex pattern evaluated the entire URL string for the @ character, failing to distinguish between an attacker using the classic domain-spoofing trick (where an @ symbol in the authority section precedes a rogue destination domain, such as http://trusted.com@malicious.com) and a legitimate application embedding an email address as a standard variable value in the query path.

To resolve this critical operational false positive, the evaluation feedback loop mandated an immediate architectural refinement of Rule 2. The regular expression was strictly scoped to evaluate only the *authority section* of the URL (demarcated strictly between the protocol scheme and the first root path slash or query question mark):

Python

```
# Refined Rule 2 Evaluation

authority_section = extracted_url.split('/')[10] if '//' in extracted_url else ''

if '@' in authority_section:

 # Trigger domain-spoofing penalty
```

By restricting the evaluation to the authority section, legitimate query-string email addresses were safely ignored, while genuine domain-spoofing attempts continued to be caught with absolute precision. This live-inbox discovery and subsequent resolution represent one of the most practically important contributions of this research, proving that heuristic rules must be mathematically bounded to real-world structural semantics.

6.5 Case Study 3: Browser Extension Content Security Policy (CSP) Interception Challenge

The third major challenge emerged during the empirical evaluation of Module 4, the Chromium browser extension. When the extension successfully intercepted a simulated phishing navigation via chrome.webNavigation.onBeforeNavigate and attempted to render the local

warning page (warning.html), the browser threw a fatal execution error, displaying a blank screen instead of the expected threat advisory.

Diagnostic inspection revealed that the failure stemmed from modern browser security restrictions enforced by the Manifest V3 Content Security Policy (CSP), restrictions that recent empirical analysis confirms substantially reduce the attack surface available to malicious or poorly secured extensions [31]. The original prototype implementation of warning.html incorporated inline JavaScript event listeners to handle user interaction buttons (such as "Go Back" to safety or "Proceed at Your Own Risk"). Manifest V3 strictly prohibits the execution of inline scripts as a foundational defence against Cross-Site Scripting (XSS) vulnerabilities.

To resolve this platform-specific dependency challenge, the extension's frontend architecture was refactored. All JavaScript logic was completely stripped from the HTML file and abstracted into a dedicated, external script file (warning.js), referenced properly via standard script tags. This structural separation fully satisfied Manifest V3 CSP compliance requirements, allowing the warning interface to render smoothly, display calculated heuristic risk scores, and process user navigation choices securely.

6.6 Cross-Engine Consistency Verification (Python vs. JavaScript Parity)

A fundamental design requirement of the "one engine, many doors" architectural paradigm was ensuring that identical threat-intelligence rules yield identical diagnostic verdicts across divergent runtime environments, specifically Python 3 on the backend and client-side JavaScript within the browser extension.

To empirically verify this cross-language parity, a standardized test array comprising four complex, edge-case URLs (incorporating combinations of raw IP addresses, puny code homographs, suspicious TLDs, and brand impersonation strings) was formulated. Each URL was passed independently through both the Python core scanner engine (core_scanner.py) and the JavaScript browser extension engine (heuristics.js).

The comparative analysis of the resulting output structures confirmed absolute logical parity. Both execution environments produced identical numeric risk scores, identical final risk tier classifications ("CLEAN," "SUSPICIOUS," or "MALICIOUS"), and identical arrays of human-readable triggered reasons for every test case. This empirical verification proves that centralized security policies can be distributed across hybrid software architectures without introducing logical drift or policy discrepancies.

6.7 Comprehensive Regression Testing and Final System State

Following the iterative resolution of the PowerPoint binary check, the live query-string anomaly, and the browser extension CSP restriction, a detailed final regression test suite was executed across the entire software artefact.

The complete test suite, comprising all eight synthetic files, three offline email archives, live IMAP batch scans, and browser extension interception triggers, was run consecutively. Every test case yielded correct, expected, and reproducible outcomes in strict accordance with the system design specifications. The empirical evaluation overall demonstrates that a lightweight, fully explainable heuristic system can achieve strong, verifiable threat detection across diverse operational entry points, providing enterprise administrators with reliable, transparent security controls.

6.8 Positioning Against Literature-Reported Benchmarks

A live benchmark against a competitor tool was outside this project's scope, but the results can still be positioned against detection rates reported in the literature reviewed in Chapter 2 - an indicative, not statistically controlled, comparison, since test corpora and conditions differ.

The heuristic engine achieved 100% accuracy with zero false positives across its synthetic suite, consistent with [16] (97.13% true-positive rate, <1.5% false positives, on a much larger dataset) and with the 97.78% accuracy [20] report for a SHAP-instrumented classifier.

Two caveats matter: the synthetic suite here is far smaller than these literature datasets, so the 100% figure reflects full coverage of a small test matrix rather than population-level accuracy, not a claim of superiority; and the comparison

uses different test conditions (static benchmarks vs. live IMAP traffic). A same-corpus comparison against a published baseline is flagged as future work (Section 7.3).

6.9 A Threats-to-Validity Perspective on the Evaluation

Framing the limitations above in the standard threats-to-validity structure used in empirical software engineering research clarifies precisely what this evaluation does and does not establish and separates genuine weaknesses from properties that are actually strengths of a deterministic design.

Internal validity: because the heuristic engine is fully deterministic, every result is exactly reproducible given the same input - there is no random initialisation or training variance to account for, which is a genuine strength relative to ML baselines. The threat is that the synthetic test files were authored by the researcher specifically to trigger known rules, which risks confirmation bias; an independently constructed or adversarial test set was outside this project's timeline but is the natural next check.

External validity: the live-testing component drew on a single Gmail account over a limited window, so the false-positive and false-negative behaviour observed may not generalise to other email providers, larger user populations, or attacker techniques not represented in either the synthetic suite or that inbox's traffic during the test period. This is the same limitation already discussed in Sections 6.8 and 7.2, restated here in standard terms.

Construct validity: the CLEAN/SUSPICIOUS/MALICIOUS categorisation is a simplification of a continuous risk space, and this evaluation treats a false positive and a false negative as equally costly when reporting accuracy. In a real deployment the two error types carry asymmetric costs - a missed phishing email is typically far more costly than an unnecessary warning - and a fuller evaluation would weight them accordingly rather than reporting a single accuracy figure.

Reliability: repeating the synthetic suite against the same code will always reproduce the same verdicts, which is straightforward to verify and is itself evidence for the explainability claims made in Chapter 4. Repeating the live-

inbox component would not reproduce identical results, since real email traffic is not static; this is an inherent property of live validation rather than a flaw in the method, and is precisely why the synthetic suite and the live test were designed to serve different, complementary purposes in this project's evaluation strategy.

6.10 Tracing the Rule Engine Against Documented Real-World Cases

As a qualitative supplement to the synthetic and live testing above, this section traces the system's 8-rule logic against independently documented real-world phishing cases from the security industry literature, rather than the project's own synthetic files. This is a manual logic trace against published case descriptions, not a code-execution experiment, and is offered as corroborating evidence alongside - not a replacement for - the structured baseline comparison identified in Section 7.3.

Case 1 - IDN homograph attack on Google News: security researcher Avi Lumelsky documented a live phishing domain, google.news, which substitutes a Cyrillic g for the Latin g to visually impersonate google.news [32]. Applying the system's Rule 5 (Punycode/IDN homograph indicator, +25 points) to this documented case would correctly flag the domain, though at 25 points alone it would fall short of the 50-point MALICIOUS threshold - a single real homograph attack, in isolation, would be scored SUSPICIOUS rather than blocked outright under this system's deliberate two-signal design.

Case 2 - brand-impersonation login pages: Netcraft's 2026 review of real phishing websites documents multiple live cases - fake Netflix, Instagram, and Booking.com login pages - each clearly characterised by being hosted on a domain that does not belong to the brand being impersonated [33]. Applying Rule 8 (brand name present in non-official domain, +30 points) to each of these documented cases would correctly flag them, again at a single-rule score below the MALICIOUS threshold on its own.

Taken together, these two independently documented, real-world cases corroborate that the individual rules fire correctly against genuine attack patterns, not only against the project's own synthetic examples. They also surface an honest observation the synthetic suite alone could not: a real attack

that relies on exactly one of these techniques in isolation would be scored SUSPICIOUS, not MALICIOUS, under the current calibration. In practice this is unlikely to be a serious gap, since Netcraft's own case descriptions show real phishing pages typically combining multiple techniques simultaneously (a homograph domain and a fake login form, for example, would additionally trigger structural or brand-related signals) - but it is a precise, evidence-based boundary of the current threshold, rather than an unexamined assumption, and is noted as a candidate refinement for future work.

CHAPTER 7 — CONCLUSIONS, RECOMMENDATIONS, AND REFLECTION

7.1 Conclusions

This research project set out to design, implement, and carefully evaluate a detailed, multi-layer threat-detection and data protection system tailored specifically for current enterprise and regional branch-office environments. Driven by the pressing operational challenge of fragmented security policies, vendor lock-in, and the widespread opacity of traditional security tools, the study successfully formulated and validated the "one engine, many doors" architectural paradigm.

By centralizing all parsing logic, cryptographic integrity verification, and heuristic rule evaluation into a unified core engine, the system demonstrated that absolute policy consistency can be maintained across highly divergent operational entry points, spanning static desktop files, offline email archives, live IMAP mailboxes, and real-time Chromium browser navigation. The empirical evaluation overall answered the central research question, proving that a transparent, rule-based heuristic scoring engine can achieve strong, verifiable threat detection without relying on the large computational overhead, training-data requirements, or black-box opacity characteristic of machine-learning alternatives.

Also, the integration of advanced structural analysis, specifically addressing macro-enabled document identification, PDF object inspection, hidden image hyperlink extraction, and QR-code ("quishing") payload decoding, directly resolved major gaps identified in current security literature regarding attachment and obfuscation coverage. The project bridges the vital gap between low-level software engineering and high-level international compliance standards, proving that lightweight, explainable technical controls can effectively satisfy the careful demands of frameworks such as ISO/IEC 27001:2022 and the UK GDPR.

7.2 Technical and Operational Limitations

While the system successfully achieved its stated aims and demonstrated strong performance across both synthetic and live evaluation phases, academic transparency requires a candid assessment of its inherent limitations:

1. **Static Heuristic Evasion: Because the detection engine relies on deterministic rule matching and static structural markers (such as the presence of vbaProject.bin or specific object dictionaries), highly sophisticated threat actors using novel, deeply obfuscated polymorphic code structures or zero-day file packing techniques could potentially bypass the static rules. The system does not incorporate dynamic behavioural sandboxing, meaning it cannot analyse runtime execution behaviour.**
2. **Bounded Live Validation Scope: The Live Mailbox Scanner module was empirically evaluated against a single, actively used personal Google Workspace (Gmail) account. While this bounded scope successfully introduced real-world operational noise and exposed critical false-positive edge cases (such as the query-string '@' symbol anomaly), a single user profile cannot mathematically represent the vast linguistic, structural, and behavioural diversity of a multinational enterprise workforce comprising thousands of distinct mailboxes.**
3. **Platform and Native-Library Dependencies:** The integration of machine-vision capabilities for QR-code decoding via pyzbar introduces external native library dependencies. In certain operating system environments (particularly Windows without pre-configured C++ redistributable runtimes), deploying the image scanning module requires additional administrative configuration, presenting a minor operational friction point for non-specialist deployment.

7.3 Recommendations for Future Work

Building upon the solid architectural foundation established in this project, several compelling avenues for future academic research and industrial development are recommended:

1. **Hybrid Threat Intelligence API Integration:** Future iterations of the core engine could incorporate asynchronous, privacy-preserving queries to external cloud threat intelligence platforms (such as the VirusTotal API or Google Safe Browsing API) as an optional secondary validation layer. This would combine the speed and explainability of local heuristics with the broad historical visibility of global reputation databases. A small-scale precedent exists: [34] describe Secure Scan, combining heuristics, a classifier, and VirusTotal lookups, though its 93.1% accuracy appears in an unreviewed preprint without dataset details, underscoring the need for independent validation first.

2. **Implementation of the Network Port-Scanning Module:** As conceptualized in the initial design methodology but deferred due to strict time constraints, a lightweight, custom Python-based network vulnerability assessment and port-scanning module should be developed and integrated into the core architecture. This would extend the system's defensive posture from ingress file and email filtering to active internal network visibility and shadow IT discovery.

3. **Expanded Enterprise Telemetry and Multi-User Logging: Deploying the live mailbox and browser extension modules across a multi-user pilot group within a controlled corporate testbed would generate valuable longitudinal data regarding administrator response times, false-positive frequencies across diverse user departments, and system resource use under high-volume enterprise loads.**

4. **Structured Baseline Benchmarking:** A same-corpus, controlled comparison against at least one published rule-based system (such as SatheeshKumar, Srinivasagan and UnniKrishnan, 2022) and one explainable machine-learning system (such as Uddin et al., 2025)

would allow the accuracy, false-positive rate, and explainability trade-offs identified in Section 6.8 to be evaluated under identical test conditions, strengthening the comparative claims made in this study.

7.4 Personal Reflection

When I started this project, I did not have a strong programming background, and that shaped almost everything about how this research actually unfolded. Tools that a Computer Science student might take for granted, setting up a terminal, navigating a code editor, or even installing a Python library, were things I had to learn from scratch. I figured them out often through trial and error, all while trying to keep the actual security logic of the project moving forward. Looking back, I think this turned out to be a useful constraint rather than just a disadvantage. It forced me to understand exactly why something worked before I trusted it, rather than just assuming it worked because the code "looked right."

A good example of this was when I had to debug the false positive on the clean PowerPoint file (detailed in Section 6.3). At first, I honestly did not understand why a completely normal administrative file was being flagged as dangerous. Working through it step by step, checking what was actually inside the file structure, and then comparing it against a file that genuinely did have a macro, taught me more about how Office files are structured than any amount of textbook reading would have.

The same was true later, and in a much more memorable way, when the live Gmail test flagged a genuine Google security email (Section 6.4). That one particular moment, realizing the bug was hiding inside my own real inbox, and not just in a test file I had built myself, is probably the single clearest lesson I am taking away from this whole project. I realized that a system that only proves itself against data you built to make it pass is not the same thing as a system that has actually been tested in the real world.

Setting up the browser extension was its own separate struggle. I had never built anything for a web browser before. The Content Security Policy issue I described in Section 6.5, where the warning page’s own script was silently blocked by Chrome’s security rules, took real patience to track down. There

was no error message telling me what was wrong; the page just quietly failed to show the right information. Getting that finally working, and seeing the warning page correctly display the blocked link, the score, and the reasons, was easily one of the most satisfying moments of the project for me.

I also learned, somewhat the hard way, that a project like this does not fail only at the level of code logic. The QR-code detection feature (Section 5.1.6) was written correctly from the start, but it silently did nothing on my own Windows laptop for a while simply because of a missing native library that had nothing to do with my Python code at all. That was a very useful reminder that a tool meant for a real administrator to install and use has to survive contact with an ordinary machine, not just a perfect development environment.

On a personal note, there were also several stretches during this project where I could not work at all because I was unwell. Looking back at my own project timeline, these gaps added up to a real amount of lost time, not just a day or two here and there. Each time I came back to the project after being sick, I had to spend hours re-reading my own earlier work just to remember exactly where I had left off. That experience taught me that keeping clear, detailed notes as you go is not optional when your health is unpredictable, it is the only thing that lets you actually pick the work back up.

If I were starting this project again, I think I would try to test against real data, like the live Gmail inbox, much earlier in the process, rather than treating it as something to do only once the “main” system was finished. The single most useful bug I found in the whole project only showed up because of that real-world test, and I only ran it late in the timeline. I would also spend less time being surprised by environment problems (like missing libraries or browser security rules) and actively build in time to expect them. In hindsight, almost every “the code should just work” moment in this project eventually needed some kind of environment-specific fix.

Overall, working on this dissertation changed how I think about what it means for security software to be trustworthy. It is not enough for a tool to be accurate in a lab; it needs to explain itself to the user. It needs to survive being tested against real, messy, unpredictable data, and it needs to keep working once it

leaves the hands of the person who built it. That is a very different standard from simply asking “does the code run,” and it is the standard I tried my best to hold this project to.

REFERENCES

[1] Federal Bureau of Investigation Internet Crime Complaint Center (IC3) (2026) *2025 Internet Crime Report*. Available at: https://www.ic3.gov/AnnualReport/Reports/2025_IC3Report.pdf (Accessed: 10 September 2026).

[2] Galadima, S.A., Shrivastava, M. and Sonowal, G. (2024) 'QR Code Phishing: A Review of the Current Attacks and Countermeasures', *Juni Khyat Journal*, 14(9-3). Available at: http://junikhyatjournal.in/no_1_Online_24/28_online_sept.pdf.

[3] Lim, S. and Oh, J. (2025) 'Navigating Privacy: A Global Comparative Analysis of Data Protection Laws', *IET Information Security*, 2025(1), 5536763. Available at: https://doi.org/10.1049/ise2/5536763.

[4] International Organization for Standardization (2022) *ISO/IEC 27001:2022 - Information security, cybersecurity and privacy protection - Information security management systems - Requirements*. Geneva: ISO. Available at: https://www.iso.org/standard/27001.html.

[5] Ali, R. (2026) 'Operationalising Information Security Management: A Procedural Framework Analysis of ISO/IEC 27001:2022 Implementation in a Financial-Technology Organisation', arXiv preprint, arXiv:2604.23230. Available at: https://arxiv.org/abs/2604.23230.

[6] Shajahan, S., George, T., Jithi, P.V., Snehamol, K.M., Krishna, S., Krishna, S. and Jagadish, N. (2025) 'Browser Extension for Phishing Website Detection Using Machine Learning', *AIJR Proceedings*, 7(5), pp. 30-37. Available at: https://doi.org/10.21467/proceedings.7.5.5.

[7] Arun, A. and Abosata, N. (2024) 'Next Generation of Phishing Attacks using AI powered Browsers', arXiv preprint, arXiv:2406.12547. Available at: https://arxiv.org/abs/2406.12547.

[8] Bac, T.N., Duy, P.T. and Pham, V.-H. (2021) 'PWDGAN: Generating Adversarial Malicious URL Examples for Deceiving Black-Box Phishing Website Detector using GANs', in *2021 IEEE International Conference on Machine Learning and Applied Network Technologies (ICMLANT)*, pp. 1-4. Available at: https://doi.org/10.1109/ICMLANT53170.2021.9690540.

[9] Sabir, B., Babar, M.A., Gaire, R. and Abuadbba, A. (2022) 'Reliability and Robustness Analysis of Machine Learning Based Phishing URL Detectors', *IEEE Transactions on Dependable and Secure Computing*, 19(6), pp. 2944-2959. Available at: https://doi.org/10.1109/TDSC.2022.3218043.

[10] Das Guptta, S., Shahriar, K.T., Alqahtani, H., Alsalman, D. and Sarker, I.H. (2022) 'Modeling Hybrid Feature-Based Phishing Websites Detection Using Machine Learning Techniques', *Annals of Data Science*, 11(1), pp. 217-242. Available at: https://doi.org/10.1007/s40745-022-00379-8.

[11] Thakur, K., Ali, M.L., Obaidat, M.A. and Kamruzzaman, A. (2023) 'A Systematic Review on Deep-Learning-Based Phishing Email Detection', *Electronics*, 12(21), 4545. Available at: https://doi.org/10.3390/electronics12214545.

[12] Kyaw, P.H., Gutierrez, J. and Ghobakhlou, A. (2024) 'A Systematic Review of Deep Learning Techniques for Phishing Email Detection', *Electronics*, 13(19), 3823. Available at: https://doi.org/10.3390/electronics13193823.

[13] SatheeshKumar, M., Srinivasagan, K.G. and UnniKrishnan, G. (2022) 'A Lightweight and Proactive Rule-Based Incremental Construction Approach to Detect Phishing Scam', *Information Technology and Management*, 23(4), pp. 271-298. Available at: https://doi.org/10.1007/s10799-021-00351-7.

[14] Tupsamudre, H., Jain, S. and Lodha, S. (2021) 'PhishMatch: A Layered Approach for Effective Detection of Phishing URLs', arXiv preprint, arXiv:2112.02226. Available at: https://arxiv.org/abs/2112.02226.

[15] Uddin, K.M.M., Biswas, N., Rikta, S.T., Nur-A-Alam, M. and Mostafiz, R. (2025) 'Explainable Machine Learning for Phishing Site Detection: A High-Efficiency Approach Using Boosting Models and SHAP', *The Journal of Engineering*, 2025(1), e70110. Available at: https://doi.org/10.1049/tje2.70110.

[16] Mia, M., Derakhshan, D. and Pritom, M.M.A. (2024) 'Can Features for Phishing URL Detection Be Trusted Across Diverse Datasets? A Case Study with Explainable AI', in *Proceedings of the 11th International Conference on Networking, Systems, and Security (NSysS '24)*, Khulna, Bangladesh, pp. 137-145. ACM. Available at: https://doi.org/10.1145/3704522.3704532.

[17] Liu, R., Lin, Y., Yu, S.Y.S., Teoh, X., Liang, Z. and Dong, J.S. (2025) 'PiMRef: Detecting and Explaining Ever-Evolving Spear Phishing Emails with Knowledge Base Invariants', arXiv preprint, arXiv:2507.15393. Available at: https://arxiv.org/abs/2507.15393.

[18] Nejati, F., Rabbani, M., Eskandarian, M., Mirani, M., Piya, G., Opushnyev, I., Ghorbani, A.A. and Dadkhah, S. (2026) 'CIC-Trap4Phish: A Unified Multi-Format Dataset for Phishing and Quishing Attachment Detection', arXiv preprint, arXiv:2602.09015. Available at: https://arxiv.org/abs/2602.09015.

[19] Chen, X., Wang, W. and Han, W. (2023) 'Malicious Office Macro Detection: Combined Features with Obfuscation and Suspicious Keywords', *Applied Sciences*, 13(22), 12101. Available at: https://doi.org/10.3390/app132212101.

[20] Rafsanjani, A.S., Kamaruddin, N.B., Rusli, H.M. and Dabbagh, M. (2023) 'QsecR: Secure QR Code Scanner According to a Novel Malicious URL Detection Framework', *IEEE Access*, 11, pp. 92523-92539. Available at: https://doi.org/10.1109/ACCESS.2023.3291811.

[21] Sonowal, G. et al. (2024) 'QR Code Phishing: A Review of the Current Attacks and Countermeasures', *Juni Khyat Journal*, 14(9-3). Available at: http://junikhyatjournal.in/no_1_Online_24/28_online_sept.pdf.

[22] Öztürk, C. and Kara, M. (2024) 'Practical Comparison of Nessus, Qualys and OpenVAS Vulnerability Scanner Tools', in *Proceedings of the 5th International Eurasian Conference on Science, Engineering and Technology (EurasianSciEnTech 2024)*, Ankara, Turkey, 26-28 June 2024, pp. 710-716.

[23] Alavizadeh, H., Jang-Jaccard, J., Enoch, S.Y., Al-Sahaf, H., Welch, I., Camtepe, S.A. and Kim, D.S. (2021) 'A Survey on Threat Situation Awareness Systems: Framework, Techniques, and Insights', arXiv preprint, arXiv:2110.15747. Available at: https://arxiv.org/abs/2110.15747.

[24] Storey, M.-A., Hoda, R., Milani, A.M.P. and Baldassarre, M.T. (2025) 'Guiding Principles for Mixed Methods Research in Software Engineering', *Empirical Software Engineering*, 30, 138. Available at: https://doi.org/10.1007/s10664-025-10629-x.

[25] Eyilmez, K., Basyurt, A.S., Stieglitz, S., Fuchss, C., Kaufhold, M.-A., Reuter, C. and Mirbabaie, M. (2022) 'A Design Science Artefact for Cyber Threat Detection and Actor Specific Communication', in *ACIS 2022 Proceedings*, paper 50, pp. 1-8. Available at: https://aisel.aisnet.org/acis2022/50/.

[26] Shajahan, S. et al. (2025) 'Browser Extension for Phishing Website Detection Using Machine Learning', *AIJR Proceedings*, 7(5), pp. 30-37. Available at: https://doi.org/10.21467/proceedings.7.5.5.

[27] Pantelaios, N. and Kapravelos, A. (2024) 'Manifest V3 Unveiled: Navigating the New Era of Browser Extensions', in *Proceedings of the Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb 2024)*. Available at: https://doi.org/10.14722/madweb.2024.23080.

[28] Valimail (n.d.) 'URL Phishing: What It Is, Real World Examples & Strategies'. Available at: https://www.valimail.com/resources/guides/guide-to-phishing/url-phishing-real-world-examples-strategies/ (Accessed: 25 September 2026).

[29] Netcraft (2026) '10 Real Examples of Phishing Websites (and How to Spot Them)'. Available at: https://www.netcraft.com/blog/10-real-examples-of-phishing-websites-screenshots (Accessed: 25 September 2026).

[30] Firdos, R. and Dangi, A. (2026) 'SecureScan: An AI-Driven Multi-Layer Framework for Malware and Phishing Detection Using Logistic Regression and Threat Intelligence Integration', arXiv preprint, arXiv:2602.10750. Available at: https://arxiv.org/abs/2602.10750.

[31] Chiew, K.L., Yong, K.S.C. and Tan, C.L. (2018) 'A survey of phishing attacks: Their types, vectors and technical approaches', *Expert Systems with Applications*, 106, pp. 1-20. Available at: https://doi.org/10.1016/j.eswa.2018.03.050.

[32] Gupta, B.B., Arachchilage, N.A.G. and Psannis, K.E. (2018) 'Defending against phishing attacks: Taxonomy of methods, current issues and future directions', *Telecommunication Systems*, 67(2), pp. 247-267. Available at: https://doi.org/10.1007/s11235-017-0334-z.

[33] Khonji, M., Iraqi, Y. and Jones, A. (2013) 'Phishing detection: A literature survey', *IEEE Communications Surveys & Tutorials*, 15(4), pp. 2091-2121. Available at: https://doi.org/10.1109/SURV.2013.032213.00009.

[34] Abu-Nimeh, S., Nappa, D., Wang, X. and Nair, S. (2007) 'A comparison of machine learning techniques for phishing detection', in *Proceedings of the Anti-Phishing Working Groups 2nd Annual eCrime Researchers Summit 2007*, Pittsburgh, Pennsylvania, USA, pp. 60-69. ACM. Available at: https://doi.org/10.1145/1299015.1299021.

[35] Fette, I., Sadeh, N. and Tomasic, A. (2007) 'Learning to detect phishing emails', in *Proceedings of the 16th International Conference on World Wide Web (WWW 2007)*, Banff, Alberta, Canada, pp. 649-656. ACM. Available at: https://doi.org/10.1145/1242572.1242660.

[36] Sahingoz, O.K., Buber, E., Demir, O. and Diri, B. (2019) 'Machine learning based phishing detection from URLs', *Expert Systems with Applications*, 117, pp. 345-357. Available at: https://doi.org/10.1016/j.eswa.2018.09.029.

[37] Li, T., Kou, G. and Peng, Y. (2020) 'Improving malicious URLs detection via feature engineering: Linear and nonlinear space transformation methods', *Information Systems*, 91, 101494. Available at: https://doi.org/10.1016/j.is.2020.101494.

[38] Almseidin, M., Abu Zuraiq, A., Al-kasassbeh, M. and Alnidami, N. (2019) 'Phishing detection based on machine learning and feature selection methods', *International Journal of Interactive Mobile Technologies*, 13(12), pp. 171-183. Available at: https://doi.org/10.3991/ijim.v13i12.11411.

[39] Dandotiya, M., Goyal, N.K., Khunteta, A. and Tiwari, B. (2026) 'Real time identification of phishing attacks through machine learning enhanced browser extensions', *Scientific Reports*, 16, 6612. Available at: https://doi.org/10.1038/s41598-026-35655-7.

[40] Filio-Aguilar, P.-D., Mil-Martínez, R. and López-García, L. (2026) 'Quishing: A sociotechnical framework for understanding QR-code phishing risks and user-centered protection strategies', *Journal of Cybersecurity and Privacy*, 6(4), 134. Available at: https://doi.org/10.3390/jcp6040134.