ExRay: Hybrid Interpretable Machine Learning for Malicious Chrome Extension Detection

Elijah Segura
Department of Computer Science
Central Michigan University
Mount Pleasant, MI, USA
segur1em@cmich.edu

Abstract—**Browser extensions present significant security risks, since they operate with elevated privileges and access to sensitive user data. Traditional rule-based detection systems suffer from high false positive rates (61% as tested. Machine learning approaches, though more accurate, typically function as black boxes that cannot explain their decisions to users. We present ExRay - a hybrid interpretable malware detection system for Chrome extensions, combining static rule-based analysis with an interpretable Random Forest classifier. Our key contribution is semantic aggregation of SHAP (SHapley Additive exPlanations) values into seven domain specific risk categories, helping to increase understanding of classifications. We curated a real-world dataset of 142 Chrome extensions (72 malicious, 70 benign) with labeling from Chrome Web Store removal flags. Our system achieves 82% cross-validation accuracy and 77% hold-out test accuracy, a 20 percentage point improvement over the rule-based approach, while reducing false positive rates by 35 percentage points. The interactive web interface enables users to explore a detailed analysis on why an uploaded or preset extension is classified as malicious or benign, and comparison between the two approaches.**

I. Introduction

Browser extensions have become ubiquitous tools for enhancing web browsing functionality. As of 2024, the Chrome Web Store hosted over 180,000 extensions with billions of cumulative downloads. However, this popularity has made extensions an attractive attack vector for malicious actors [1]. Malicious extensions can steal credentials, inject advertisements, track user behavior, and exfiltrate sensitive data. All while appearing as legitimate productivity tools.

A. Problem Statement

The detection of malicious browser extensions presents a challenging security problem that traditional approaches fail to adequately address.

Rule-Based Systems: Traditional signature and heuristic-based detection methods suffer from high false positive rates. In our experiments, our rule-based analysis incorrectly flagged 61% of benign as extensions as potentially malicious, insufficient for user trust and reliability. These type of systems struggle to distinguish between legitimate uses of APIs (e.g. password managers requiring credential access) and malicious abuse of the same capabilities.

Black-Box Machine Learning: While ML-based approaches achieve higher accuracy, they typically operate as opaque classifiers that provide probability scores without explanation [2]. If a security analyst cannot understand why an extension was flagged, it introduces difficultly for verifying decisions, prioritizing responses, or improving detection rules. This lack of interpretability is particularly problematic in security contexts where professionals must justify blocking decisions [9].

The Interpretability Gap: A fundamental tensions exists between detection accuracy and explainability. Deep learning approaches such as neural networks for JavaScript analysis [3] achieve strong performance but sacrifice interpretability. Conversely, simple rule-based systems are transparent but miss sophisticated threats.

B. Our Contribution

We address this gap with a hybrid interpretable malware detection system that provides both accurate classification and human-understandable explanations. Our primary contribution is **semantic aggregation of SHAP values**, which transforms raw feature importances into meaningful risk categories that security practitioners can understand and act upon.

By transforming raw features importance into meaningful risk categories, users can understand and act upon predictions presented by the system.

Instead of presenting feature names like:

- `perm_scripting`: 0.15
- `tfidf_eval`: 0.12
- `perm_webRequest`: 0.08

Our system aggregates these into interpretable categories:

- **Permissions Risk (26% impact)**: scripting, webRequest, cookies
- **Code Behavior (18% impact)**: eval usage, dynamic function creation
- **Network Activity (11% impact)**: fetch calls, XMLHttpRequest

We can summarize our work with 5 major contributions:

1) A **hybrid analysis architecture** combining separate transparent rule-based scoring with accurate ML classification
2) **SHAP-based category aggregation** that groups feature importance's into seven semantic risk categories
3) A **curated dataset** of 142 Chrome extensions with ground-truth malware labels
4) An **interactive web interface** enabling exploration of risk factors with detailed explanations

5) **Comprehensive evaluation** demonstrating 20+ percentage point accuracy improvement over rule-based systems

II. Related Work

A. Browser Extension Security

Sanchez-Rola et al. [4] conducted one of the first comprehensive security analyses of browser extension resource control policies, identifying vulnerabilities in how extensions manage access to web content. Their work highlighted the inadequacy of permission-based security models, motivating our multi-factor approach that examines both manifest permissions and actual code behavior.

Kim and Lee [1] demonstrated that browser extensions can be exploited for privilege escalation attacks, with malicious extensions leveraging legitimate API access to compromise browser security. Their findings emphasizes the importance of detecting not just obviously malicious patterns, but also subtle capability combinations that enable attacks. Our permission-risk scoring incorporates insights from their attack taxonomy.

Laperdrix et al. [5] showed that browser extensions can be fingerprinted through their injected style sheets, enabling tracking across sessions. While their work focuses on privacy implications, it demonstrates the extensive capabilities available to extensions, those that malicious actors can abuse.

B. Machine Learning for Malware Detection

The application of machine learning to malware detection has received extensive attention. Gaber et al. [6] provide a systematic literature review covering 290 studies, identifying feature extraction, model selection, and evaluation methodology as critical success factors. Our work focuses on their identified gap in interpretability by incorporating SHAP-based explanations.

Stokes et al. [3] developed ScriptNet, a neural network approach for malicious JavaScript detection using LSTMs. While their model achieves strong performance, it operates as a black box. Our Random Forest approach sacrifices some potential accuracy for interpretability, enabling security analysts to understand and verify classification decisions.

Gobbi and Kinder [7] presented GENIE for detecting malicious npm packages using semantic analysis. Their work on the JavaScript ecosystem is similar to ours, though we focus specifically on browser extensions with their unique manifest-based permission model. Both approaches recognize the importance of understanding code behavior beyond simple pattern matching.

C. Explainable AI for Security

The need for interpretable ML in security contexts has been well established. Lin and Chang [2] surveyed interpretation methods for ML-based malware detection, categorizing approaches by explanation granularity and identifying SHAP as particularly suitable for feature-based models. Our category aggregation extends their framework by grouping SHAP values into domain-specific risk categories.

To et al. [8] developed MalDEX, an explainable malware detection system using ensemble learning with SHAP visualizations. Our work differs by focusing on browser extensions rather than executables, and by introducing semantic category aggregation that produces more actionable explanations.

Baghirov [9] examined advanced ML and interpretability for Windows malware detection, emphasizing the importance of human-understandable explanations for security operations. Our approach extends these insights to the browser extension domain, where the permission model provides natural semantic categories for explanation.

D. Hybrid Detection Approaches

Hybrid systems combining rules and ML have shown promise across security domains. Zeeshan and Amjad [10] demonstrated effective hybrid malware classification for PDFs, achieving improved accuracy over single-method approaches. Our architecture similarly leverages the complementary strengths of rule-based transparency and ML accuracy, using rules for detailed threat scoring, while ML provides overall classification.

III. Methodology and System Design

A. System Architecture

Our system follows a pipeline architecture as shown in Figure 1. User-uploaded extensions pass through feature extraction, dual analysis (rules + ML), and explainability modules before results are displayed in an interactive interface.

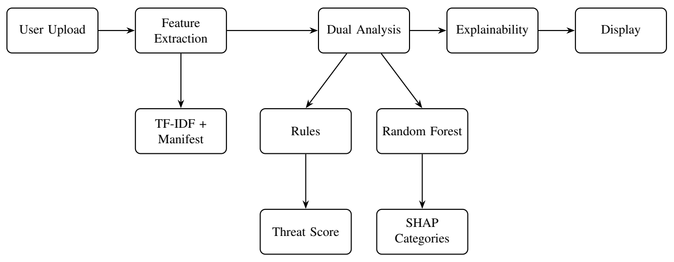


Fig. 1: High-level system architecture showing dual analysis paths

B. Feature Extraction

We extract 1,019 features from each extension, combining static code analysis with manifest inspection:

TF-IDF Code Features (1,000): We apply Term Frequency-Inverse Document Frequency vectorization to all JavaScript files in an extension, using a vocabulary of 1,000 terms. This captures code patterns including API calls, variable names, and suspicious strings associated with malicious behavior.

Manifest Permission Features (19): Binary features encode the presence of permissions:

- High-risk: `scripting`, `webRequest`, `management`, `debugger`
- Medium-risk: `tabs`, `cookies`, `downloads`, `nativeMessaging`

- Low-risk: `storage`, `notifications`, `contextMenus`
- Host access: `<all_urls>`, broad host patterns
- CSP bypass: `unsafe-eval` in Content Security Policy

C. Rule-Based Threat Scoring

Our rule engine analyzes extensions using Abstract Syntax Tree (AST) parsing of JavaScript code via the Esprima library. Findings are scored using a weighted saturation system:

$$\text{Score}_{\text{raw}} = \sum_{c \in \text{categories}} \min\left(\sum_{f \in c} w_f, S_c\right) \quad (1)$$

where w_f is the weight for finding f and S_c is the saturation limit for category c. Saturation prevents a single category from dominating the score. Table I shows our scoring weights.

TABLE I: Threat Scoring Weights by Category

Category	Weight	Saturation
Obfuscation	25	100
CSP Bypass	20	40
Suspicious URL	20	100
Dangerous DOM	15	60
Risky API	15	100
Permissions	10	100
Data Exfiltration	10	50

To prevent large extensions from being unfairly penalized, we apply logarithmic size normalization:

$$\text{Score}_{\text{norm}} = \text{Score}_{\text{raw}} \times \frac{1}{\log_{10}(1 + n_{\text{files}})} \quad (2)$$

D. Machine Learning Classification

We train a Random Forest classifier with 100 trees using class-balanced weights to handle dataset imbalance. The model predicts malicious probability $P(y = 1|x)$ for feature vector x.

Model Configuration:

- Algorithm: Random Forest (scikit-learn)
- Trees: 100 estimators
- Class weights: Balanced
- Features: 1,019 (TF-IDF + manifest)

E. SHAP-Based Category Aggregation

One key innovation is semantic aggregation of SHAP values. We use TreeExplainer with 50 background samples to compute feature contributions, then aggregate into seven risk categories based on domain knowledge: 1

The `GetCategory` function maps features to categories using pattern matching:

- `perm_*` → Permissions
- `unsafe_eval` → Security
- `eval`, `innerHTML` → Behavior
- `http*`, `url*` → Network
- `document.*` → DOM
- `storage`, `cookie` → Storage
- Other TF-IDF terms → Code Patterns

Algorithm 1 SHAP Category Aggregation

```
Require: Feature vector $x$, SHAP values $\phi$, feature names $F$
Ensure: Categorized risk factors $R$
1:  $C \leftarrow$ {Permissions, Security, Behavior, Network,
        DOM, Storage, Patterns}
2:  for each category $c \in C$ do
3:      $R[c] \leftarrow \{\text{impact} : 0, \text{features} : []\}$
4:  end for
5:  for each feature $i$ with $\phi_i \neq 0$ do
6:      $c \leftarrow \textsc{GetCategory}(F[i])$
7:      $R[c].\text{impact} \leftarrow R[c].\text{impact} + |\phi_i|$    ▷ Sum absolute impact
8:      $R[c].\text{features.append}(\{\text{name} : F[i], \text{imp} : \phi_i\})$
9:  end for
10: return $R$ sorted by impact
```

F. Confidence-Based Filtering

To reduce false positives, we implement confidence scoring that considers the context:

$$\text{Confidence} = C_{\text{base}} \times M_{\text{path}} \times M_{\text{safety}} \quad (3)$$

where C_{base} is the base confidence for the finding type, M_{path} reduces confidence for test/vendor files, and M_{safety} reduces confidence when safety indicators (try/catch, sanitization) are present.

Findings are filtered by severity-specific thresholds:

- Critical: 0.4 (keeps most findings)
- High: 0.5
- Medium: 0.6
- Low: 0.7 (higher bar for minor findings)

G. Interactive Web Interface

We provide a Flask-based web interface enabling:

- Extension upload via ZIP file or preset selection 2a
- Visual threat score scale with severity coloring 2b
- Expandable risk category exploration 2c
- Analysis breakdown and full JSON export 2d
- Detailed threat score breakdown and analysis 3a
- Per-finding confidence scores and descriptions with specific locations 3b
- Detailed findings with specific code snippets 3c

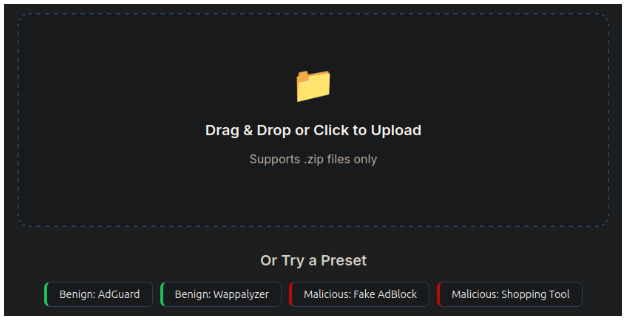
IV. Evaluation

A. Dataset

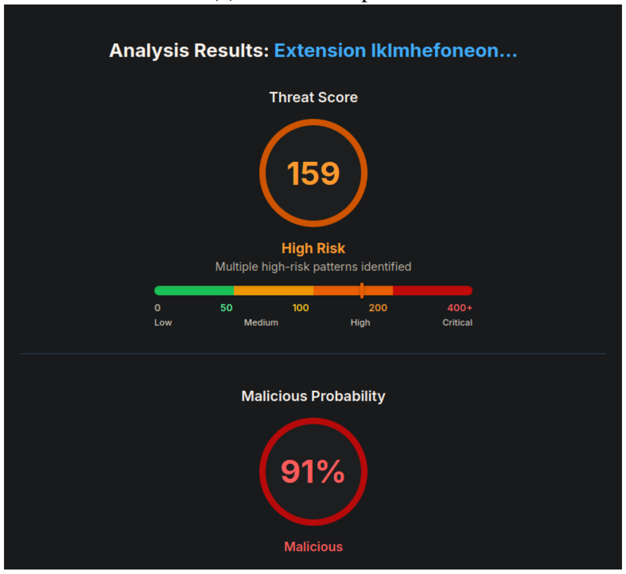
We curated a dataset of 142 Chrome extensions with ground-truth labels:

Malicious Extensions (72): Collected via Chrome-Stats API with criteria `obsolete=Exists` and `obsoleteReason=''malware''`. These extensions were flagged and removed by Google for malicious behavior, providing reliable labels.

Benign Extensions (70): Collected with criteria `obsolete=NotExists` and



(a) Extension upload


Analysis Results: Extension lklmhefoneon...

Threat Score

159

High Risk

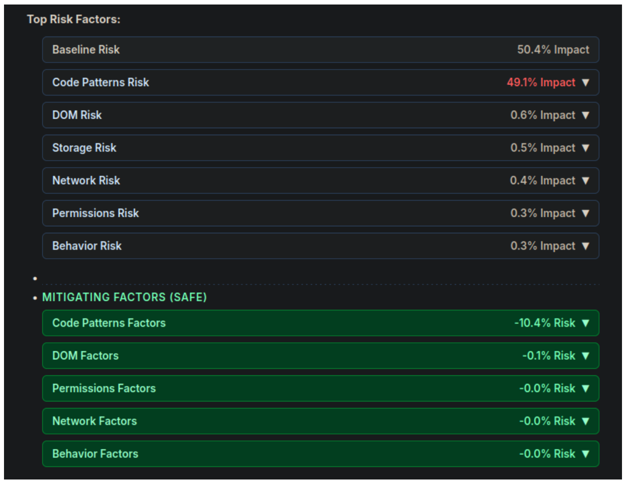
Multiple high-risk patterns identified

Malicious Probability

91%

Malicious

(b) Threat score scale

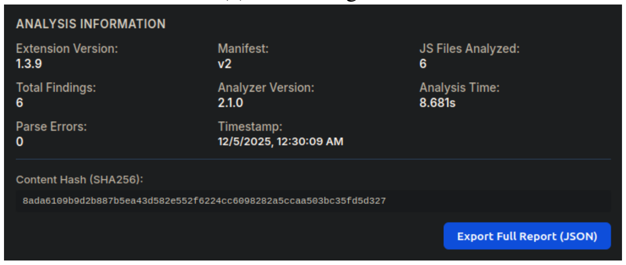

Top Risk Factors:

Baseline Risk	50.4% Impact
Code Patterns Risk	49.1% Impact ▼
DOM Risk	0.6% Impact ▼
Storage Risk	0.5% Impact ▼
Network Risk	0.4% Impact ▼
Permissions Risk	0.3% Impact ▼
Behavior Risk	0.3% Impact ▼

MITIGATING FACTORS (SAFE)

Code Patterns Factors	-10.4% Risk ▼
DOM Factors	-0.1% Risk ▼
Permissions Factors	-0.0% Risk ▼
Network Factors	-0.0% Risk ▼
Behavior Factors	-0.0% Risk ▼

(c) Risk categories


ANALYSIS INFORMATION

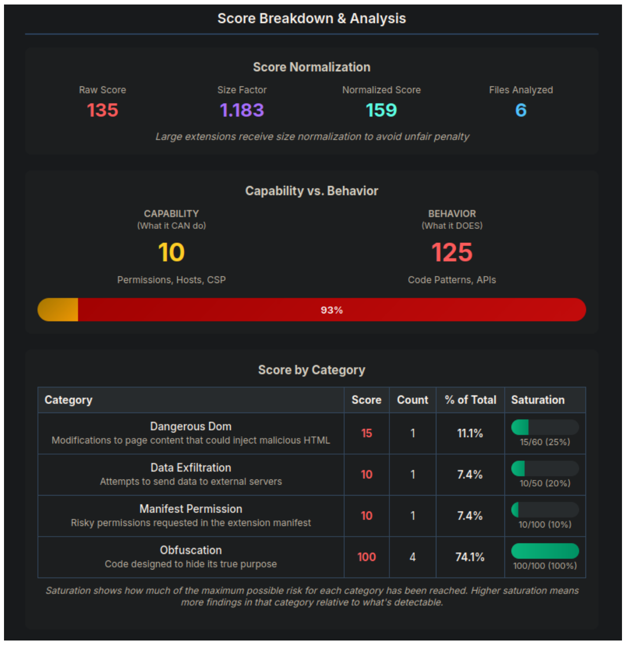
Extension Version: 1.3.9	Manifest: v2	JS Files Analyzed: 6
Total Findings: 6	Analyzer Version: 2.1.0	Analysis Time: 8.681s
Parse Errors: 0	Timestamp: 12/5/2025, 12:30:09 AM	

Content Hash (SHA256):

```
8ada6109b9d2b887b5ea43d582e552f6224cc6098282a5ccaa503bc35fd5d327
```

(d) Analysis breakdown

Fig. 2: Interface Overview (Part 1): Initial upload and high-level risk assessment.


Score Breakdown & Analysis

Score Normalization

Raw Score	Size Factor	Normalized Score	Files Analyzed
135	1.183	159	6

Large extensions receive size normalization to avoid unfair penalty

Capability vs. Behavior

CAPABILITY (What it CAN do)	BEHAVIOR (What it DOES)
10	125
Permissions, Hosts, CSP	Code Patterns, APIs

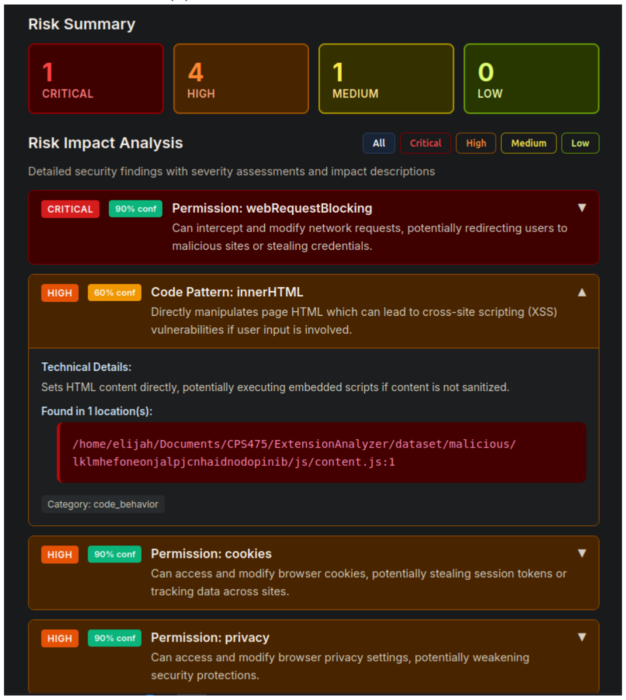
93%

Score by Category

Category	Score	Count	% of Total	Saturation
Dangerous Dom Modifications to page content that could inject malicious HTML	15	1	11.1%	15/60 (25%)
Data Exfiltration Attempts to send data to external servers	10	1	7.4%	10/50 (20%)
Manifest Permission Risky permissions requested in the extension manifest	10	1	7.4%	10/100 (10%)
Obfuscation Code designed to hide its true purpose	100	4	74.1%	100/100 (100%)

Saturation shows how much of the maximum possible risk for each category has been reached. Higher saturation means more findings in that category relative to what's detectable.

(a) Detailed threat breakdown


Risk Summary

1 CRITICAL	4 HIGH	1 MEDIUM	0 LOW

Risk Impact Analysis

Detailed security findings with severity assessments and impact descriptions

CRITICAL 90% conf **Permission: webRequestBlocking** ▼
Can intercept and modify network requests, potentially redirecting users to malicious sites or stealing credentials.

HIGH 60% conf **Code Pattern: innerHTML** ▲
Directly manipulates page HTML which can lead to cross-site scripting (XSS) vulnerabilities if user input is involved.

Technical Details:
Sets HTML content directly, potentially executing embedded scripts if content is not sanitized.

Found in 1 location(s):

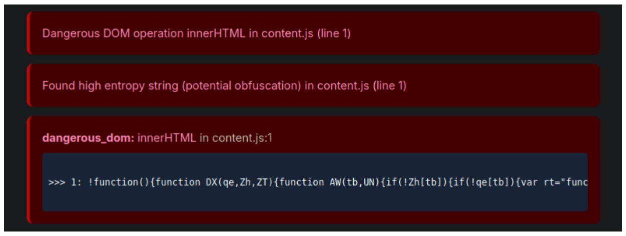
```
/home/elijah/Documents/CPS475/ExtensionAnalyzer/dataset/malicious/
lklmhefoneonjalpjcnhaidnodopinib/js/content.js:1
```

Category: code_behavior

HIGH 90% conf **Permission: cookies** ▼
Can access and modify browser cookies, potentially stealing session tokens or tracking data across sites.

HIGH 90% conf **Permission: privacy** ▼
Can access and modify browser privacy settings, potentially weakening security protections.

(b) Confidence scores


Dangerous DOM operation innerHTML in content.js (line 1)

Found high entropy string (potential obfuscation) in content.js (line 1)

dangerous_dom: innerHTML in content.js:1

```
>>> 1: !function(){function DX(qe,Zh,ZT){function AW(tb,UN){if(!Zh[tb]){if(!qe[tb]){var rt="func
```

(c) Specific findings with code

Fig. 3: Interface Overview (Part 2): Detailed scoring and code-level findings.

`isTrustedPublisher=Exists`. Active extensions from verified publishers represent legitimate software.

Table II summarizes the dataset statistics.

TABLE II: Dataset Statistics

Metric	Benign	Malicious
Extensions	70	72
Total Files	15,752	4,203
Avg JS Files/Ext	225	58

B. Evaluation Methodology

We employ two complementary evaluation approaches:

5-Fold Stratified Cross-Validation: Provides honest performance estimates with variance measurement across the full dataset.

70/30 Train-Test Split: Simulates real-world deployment where the model is trained on historical data and tested on unseen extensions.

C. Cross-Validation Results

Table III presents 5-fold cross-validation results.

TABLE III: 5-Fold Cross-Validation Results (n=142)

Metric	Mean	Std Dev ($\pm 2\sigma$)
Accuracy	82.36%	12.57%
Precision	82.24%	13.10%
Recall	83.33%	13.68%
F1-Score	82.70%	12.43%

Per-fold accuracy: [93.10%, 75.86%, 78.57%, 85.71%, 78.57%]. The variance reflects the challenge of small-dataset evaluation, which motivated our reporting of confidence percentages.

D. Hold-Out Test Results

Table IV shows performance on the 30% held-out test set (43 extensions).

TABLE IV: Hold-Out Test Results (43)

Metric	Score
Accuracy	76.74%
Precision	75.00%
Recall	81.82%
F1-Score	78.26%
False Positive Rate	28.57%
False Negative Rate	18.18%

The confusion matrix shows 15 true negatives, 18 true positives, 6 false positives, and 4 false negatives. The model prioritizes recall (identifying malicious extensions) over precision, which is more appropriate for a security screening tool.

E. Comparison with Rule-Based System

Table V compares our ML approach against the rule-based baseline.

The most significant improvement is in false positive rate reduction. Rules flagged 61% of benign extensions, which is impractical. Our ML approach reduces this to 29%.

TABLE V: ML vs Rule-Based Comparison

Metric	Rules	ML	Improvement
Accuracy	57.43%	76.74%	+19.3 pp
Precision	56.52%	75.00%	+18.5 pp
Recall	75.00%	81.82%	+6.8 pp
FPR	61.22%	28.57%	-32.7 pp

F. Interpretability Assessment

Our category aggregation produces explanations that map directly to security concepts:

Example Output for Malicious Extension:

Permissions (32% impact)
- `scripting`: Execute arbitrary JS
- `webRequest`: Intercept traffic

Behavior (28% impact)
- `eval`: Dynamic code execution
- `atob`: Base64 decoding (obfuscation)

Network (15% impact)
- `fetch`: External communication

This format allows users to quickly understand the specific risk factors that contribute to classification.

G. Limitations

Overfitting: We observed a 23 percentage point gap between training accuracy (100%) and test accuracy (77%), indicating some overfitting. With 1,019 features and 142 samples, our feature-to-sample ratio exceeds typical recommendations.

Dataset Size: Our dataset of 142 extensions, while carefully curated, limits generalization. Cross-validation variance of ±12% reflects this limitation.

Temporal Bias: Malicious extensions in our dataset are historical (already removed), potentially not representing emerging attack techniques.

V. Conclusion

We presented a hybrid interpretable malware detection system for Chrome extensions that bridges the gap between detection accuracy and explainability. By aggregating SHAP values into semantic risk categories, we enable users or security practitioners to understand classification decisions without sacrificing the accuracy benefits of machine learning.

Our system achieves 77-82% accuracy, representing a 20+ percentage point improvement over rule-based approaches while reducing false positive rates by over 30 percentage points. The interactive interface contributes to usability, in-depth analysis, and comparison between the ML and rule-based approaches.

Future work includes expanding the dataset to 200+ extensions, implementing deep learning approaches with attention-based explanations, and conducting user studies to evaluate explanation quality.

References

[1] Y. M. Kim and B. Lee, "Extending a hand to attackers: Browser privilege escalation attacks via extensions," in *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023, pp. 7055–7071.

[2] Y. Lin and X. Chang, "Towards interpreting ml-based automated malware detection models: a survey," 2021.

[3] J. W. Stokes, R. Agrawal, G. McDonald, and M. Hausknecht, "Scriptnet: Neural static analysis for malicious javascript detection," in *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, 2019, pp. 1–8.

[4] I. Sanchez-Rola, I. Santos, and D. Balzarotti, "Extension breakdown: Security analysis of browsers extension resources control policies," in *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017, pp. 679–694.

[5] P. Laperdrix, O. Starov, Q. Chen, A. Kapravelos, and N. Nikiforakis, "Fingerprinting in style: Detecting browser extensions via injected style sheets," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, pp. 2507–2524.

[6] M. Gaber, M. Ahmed, and H. Janicke, "Malware detection with artificial intelligence: A systematic literature review," *ACM Computing Surveys*, vol. 56, 2023.

[7] M. F. Gobbi and J. Kinder, "Genie: Guarding the npm ecosystem with semantic malware detection," in *2024 IEEE Secure Development Conference (SecDev)*, 2024, pp. 117–128.

[8] T.-N. To, H. D. Hoang, P. T. Duy, and V.-H. Pham, "Maldex: An explainable malware detection system based on ensemble learning," in *2023 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*, 2023, pp. 1–6.

[9] E. Baghirov, "Advanced machine learning and interpretability for windows malware detection," *Journal of Modern Technology and Engineering*, vol. 9, no. 3, pp. 165–177, 2024.

[10] S. M. Zeeshan Javed and F. Amjad, "Unveiling hidden threats in pdfs with hybrid malware classification," in *2024 IEEE 30th International Conference on Telecommunications (ICT)*, 2024, pp. 01–06.