

**FROM EVIDENCE TO REMEDIATION: AUTOMATED FORENSIC PIPELINES
FOR PROACTIVE CYBERTHREATS INTERVENTION**

A Dissertation
Presented to
The Academic Faculty

By

Mingxuan Yao

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering
Department of Electrical and Computer Engineering

Georgia Institute of Technology

May 2026

© Mingxuan Yao 2026

FROM EVIDENCE TO REMEDIATION: AUTOMATED FORENSIC PIPELINES FOR PROACTIVE CYBERTHREATS INTERVENTION

Thesis committee:

Dr. Brendan Saltaformaggio
The School of Electrical and Computer
Engineering
Georgia Institute of Technology

Dr. Frank Li
The School of Electrical and Computer
Engineering
Georgia Institute of Technology

Dr. Saman Zonouz
The School of Electrical and Computer
Engineering
Georgia Institute of Technology

Dr. Giovanni Vigna
Department of Computer Science
University of California in Santa Barbara

Dr. Christopher Kruegel
Department of Computer Science
University of California in Santa Barbara

Date approved: April 20, 2026

When life gives you lemonade, make lemons. Life will be all like, 'What?!'

Phil Dunphy, Modern Family

To my parents, Kexia and Duochang, for their endless love and support.
And to my beloved Coco and Yuki, for keeping me company along the way.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest appreciation to my advisor, Professor Brendan Saltaformaggio, for his unwavering guidance throughout my doctoral journey. Brendan has consistently steered me toward important research problems with real-world impact. Through countless cycles of challenge, refinement, and pushback, he cultivated in me the critical thinking and academic writing skills essential to becoming an independent researcher. His generous support sustained me through the long course of this endeavor, and I feel truly fortunate to have had him as my advisor.

I am also deeply grateful to Professor Giovanni Vigna and Professor Christopher Kruegel, who hosted me as a visiting researcher at the University of California, Santa Barbara for six months in late 2025. Their collaborative spirit and infectious passion for pursuing bold research ideas have left a lasting impression on me and will continue to inspire my future work.

I would like to thank Professor Saman Zonouz and Professor Frank Li for serving on my thesis committee. Their insightful feedback has been invaluable in strengthening this thesis.

The completion of this thesis would not have been possible without the contributions of my collaborators. I would like to extend my sincere thanks to Ranjita Pai Sridhar, Jonathan Fuller, Runze Zhang, and Haichuan Xu. I am also grateful to my industry partners, Netskope and Maltego, for their collaboration and support.

Last but not least, I would like to express my deepest gratitude to my parents for their unconditional support and encouragement to pursue my interests.

TABLE OF CONTENTS

Acknowledgments	v
List of Tables	xi
List of Figures	xiii
Summary	xv
Chapter 1: Introduction	1
1.1 Problem Statement	1
1.2 Thesis Contributions	3
1.3 Thesis Overview	4
1.3.1 MARSEA: Hiding in Plain Sight: An Empirical Study of Web Application Abuse in Malware	5
1.3.2 VADER: Proactive Detection and Remediation of Dead Drop Resolver Malware	5
1.3.3 COCO: Pulling the Strings: Proactive Forensics on Blockchain Fraud	6
Chapter 2: Hiding in Plain Sight: An Empirical Study of Web Application Abuse in Malware	7
2.1 Introduction	7
2.2 Challenges and Motivation	9

2.3	Methodology	12
2.3.1	Concolic Exploration Strategy	12
2.3.2	Web App and Vectors Identification	13
2.3.3	Session Reconstruction	17
2.3.4	Web App Modeling	18
2.4	Validating My Techniques	20
2.4.1	Vector Identification	20
2.4.2	Identities & Assets Attribution	22
2.5	WAE Malware Study Insights	23
2.5.1	WAE Malware Prevalence	23
2.5.2	WAE Malware Capabilities	25
2.5.3	Collaboration Evaluation	27
2.5.4	WAE Malware Engagement Data	29
2.5.5	Novel Collaboration	32
2.6	Related Work	36
2.7	Discussion and Limitations	38
2.8	Conclusion	38

Chapter 3: VADER: Proactive Detection and Remediation of Dead Drop Resolver Malware	39
3.1 Introduction	39
3.2 Overview	43
3.2.1 Running Example - The Mudrop Malware	43
3.2.2 Proactive Dead Drop Discovery	45

3.2.3	VADER in Practice	45
3.3	Methodology	46
3.3.1	DDR Logic Localization	47
3.3.2	De-Manipulation IO Boundary Isolation	48
3.3.3	De-Manipulation Recipe Identification	53
3.4	Post-Deployment Findings	58
3.4.1	DDR-Enabling Web Apps	58
3.4.2	VADER-Enabled Proactive Discovery	60
3.5	Pre-Deployment Evaluation	63
3.5.1	DDR Logic Localization	63
3.5.2	Validating Symbolic Expression Matching	64
3.5.3	De-Manipulation Recipe Identification	68
3.5.4	Case Studies	68
3.6	Discussion	71
3.7	Related Work	72
3.8	Conclusion	74
Chapter 4: Pulling the Strings: Proactive Forensics on Blockchain Fraud		75
4.1	Introduction	75
4.2	Motivation	79
4.2.1	Background	79
4.2.2	Preliminary Forensics Investigation	80
4.3	Design	85

4.3.1	Associated Contracts Recovery	86
4.3.2	Recipient Provenance Analysis	90
4.3.3	Capability Attribution Analysis	91
4.4	Case Studies	95
4.4.1	Case Study 1: Contracts Shared By DCWs	95
4.4.2	Case Study 2: Abuse Of CREATE2	95
4.5	Evaluation	96
4.5.1	Post Deployment Dataset Highlights	96
4.5.2	Impact Of DCW	97
4.5.3	Drill Down Into Fraud Contracts	102
4.6	Validation	103
4.6.1	Associated Contracts & Recipients	105
4.6.2	Dynamically Resolved Recipients	106
4.6.3	Capability Attribution	107
4.7	Discussion	108
4.8	Related Work	109
4.8.1	Fraud Detection On Blockchain	109
4.8.2	Smart Contract Vulnerability	110
4.9	Conclusion	111
Chapter 5: Conclusion and Future Work		112
5.1	Conclusion	112
5.2	Future Work	112

5.2.1	Detection of Malicious Traffic Targeting Web Applications	112
5.2.2	Emerging Threats in AI-Enabled Cybercrime	113
5.2.3	A Unified Science of Proactive Cybercrime Remediation	113
Appendices		115
	Appendix A: MARSEA Supplementary Material	116
	Appendix B: VADER Supplementary Material	120
	Appendix C: CoCo Supplementary Material	122
References		155
Vita		156

LIST OF TABLES

2.1	Abuse Vectors And Associated Rules.	15
2.2	Evaluation Of Abuse Vector Identification And Identity and Assets Attribution.	21
2.3	RQ1 Malware Reliance On WAE And Year Over Year (YoY) Increase Of WAE Adoption.	24
2.4	WAE Vector Landscape.	26
2.5	Collaboration Evaluation By Measuring RD And PW of Live Identity (ID) and Asset (A).	28
2.6	WAE Engagement Information.	30
2.7	Reporting To Web App Providers.	32
2.8	Multi-Stage Abuse Chain.	35
2.9	WAE Malware Migration.	36
3.1	VADER's Mudrop Proactive Discovery Results.	44
3.2	Common Data Manipulation Algorithms.	48
3.3	Distribution Of DDR Malware Found In Our Study.	57
3.4	Applying Recipes On Network Traffic Dataset To Proactively Discover Dead Drops	61
3.5	De-Manipulation Recipes Identified.	62
3.6	Validating DDR Logic Localization.	65

3.7	Validation of Symbolic Expressions Produced By VADER To Assess Functional Equivalence Across De-Manipulation Algorithm Implementations.	66
3.8	De-Manipulation Recipe Identification.	68
3.9	VADER's Banload Analysis.	70
4.1	Recovered Evidence From COCO's Forensics Investigation.	83
4.2	Fraud Contract Capabilities, Opcode, Semantic Models, And Mitigation Strategies.	92
4.3	Top 10 DCWS That Made Most Illicit Profit Through Fraud Contracts.	98
4.4	Top 15 Fraud Contracts Identified By COCO That Make Highest Illicit Profits.	101
4.5	CoCo'S VALIDATION.	104
A.1	Web Apps And Available Information	116
C.1	Map Abbreviation To Full Forms.	125

LIST OF FIGURES

1.1	Interconnection of The Three Components of My Forensics Framework. . .	2
2.1	Graph Automata Model To Guide Session Reconstruction From Interleaving Execution States.	19
3.1	Razy’s [100] DDR workflow. ❶: The malware author <i>manipulates</i> the C&C server address and <i>posts</i> it on X.com as a dead drop. ❷: Razy executes on the victim systems and <i>fetches</i> the dead drop. ❸: Razy <i>resolves</i> the dead drop content to connect to the C&C server.	40
3.2	DDR Logic Localization.	46
3.3	<i>Mudrop</i> ’s Decoder IO Boundary Isolation.	51
3.4	Symbolic Expression Matching For <i>Mudrop</i>	55
3.5	Constraint Chaining For Crypto Identification.	56
4.1	Flagged fraud contract.	77
4.2	Fraud contract in the real world.	81
4.3	Recipient investigation of C-98f804.	88
4.4	Distribution of associated contracts and illicit ETH profits across DCWs. . .	97
A.1	Late Response From Discord.	117
A.2	Confirmation By Discord.	118
A.3	Response From GitHub.	118

A.4	Response From MediaFire.	119
C.1	Scammers post the fraudulent message on forum.	122
C.2	Scammers post the fraudulent message on X.com.	123
C.3	Collaborating with FBI.	124

SUMMARY

Contemporary cyber forensics operates in a reactive mode: investigators collect evidence after an incident is reported and pursue remediation one case at a time. In this thesis, I focus on a novel cyber forensic science that extracts comprehensive evidence from a single reported incident and translate that evidence into remediation strategies that scale across the entire cyberthreat campaign.

I begin with MARSEA [1], the first large-scale empirical study of malware that abuses web applications as command-and-control infrastructure. MARSEA uses concolic analysis to automatically identify which web applications a malware sample abuses and what vectors it exploits, then employs session reconstruction and web application modeling to link extracted evidence to active malicious personas on those platforms. Deployed on 10,000 malware samples, MARSEA revealed 893 Web App-Engaged malware in 97 families abusing 29 web applications, and through collaboration with service providers, enabled the takedown of 50% of identified malicious content.

MARSEA's analysis exposed a particularly elusive subclass of web application abuse: Dead Drop Resolver malware, which encodes Command and Control server addresses as hidden payloads within benign web application content. Because these payloads are manipulated (i.e., encoded or encrypted) before being posted, the evidence extraction techniques in MARSEA alone cannot recover the full scope of the threat. My next system, VADER [2], addresses this gap by localizing the dead drop resolution logic within malware binaries, isolating the de-manipulation boundary, and extracting the underlying de-manipulation recipes that govern how dead drops are encoded. Once extracted, these recipes enable web application providers to proactively scan their platforms for unknown dead drops—shifting remediation from reactive takedown to forward-looking prevention. Applied to 100,000 malware samples, VADER identified 8,906 Dead Drop Resolver malware samples, extracted seven distinct de-manipulation recipes, and uncovered 57.1%

additional dead drops across 11 platforms, enabling the removal of 94.4% of proactively discovered dead drops and disrupting over 6,674 active malware samples.

Extending this evidence-to-remediation paradigm beyond malware, I present COCO [3], the first automated forensic analysis pipeline for blockchain-based financial fraud. Where existing defenses stop at flagging individual fraudulent contracts, COCO combines transaction-level forensics with symbolic analysis to trace and attribute entire fraud campaigns—recovering all associated fraudulent contracts from a single fraud report, investigating recipient provenance, and attributing fraudulent capabilities to the Deceptive Creator Wallets orchestrating the campaigns. Applied to 157 confirmed fraud contracts, COCO uncovered 1,283,198 associated contracts linked to 91 Deceptive Creator Wallets, responsible for approximately \$2.09 billion in illicit profits traced back to September 2017, leading to direct collaboration with the U.S. FBI and Etherscan to pursue remediation.

Together, these three systems establish a generalizable framework for proactive cyber forensics: given a single incident report, automatically extract comprehensive forensic evidence and generate remediation strategies that neutralize threats at platform and ecosystem scale.

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

The modern sophisticated cyberthreats outpace the capacity of defenders to respond. A central tension underlies this crisis: traditional cyber forensics remains overwhelmingly *reactive*. Investigators collect evidence after an incident is reported, present findings for a single case, and leave remediation to be pursued one case at a time. This reactive posture is fundamentally mismatched with the scale and persistence of modern threats, where adversaries operate across entire platforms and can trivially redeploy after any isolated takedown.

This thesis poses a single research question: *How can cyber forensics be transformed from a reactive, case-by-case discipline into a proactive science that automatically extracts comprehensive evidence from reported incidents and generates remediation strategies that scale across entire platforms and ecosystems?* I pursue this question across two major cybercrime domains that exemplify the reactive gap. Figure 1.1 illustrates these three components in relation to their respective subjects of investigation, demonstrating how the technologies uncovered in each phase influenced the development of subsequent components.

Malware-driven web application abuse. Malware authors have increasingly abandoned dedicated attacker-controlled infrastructure in favor of legitimate, widely-used web applications platforms such as Google Drive, GitHub, Discord, Dropbox, and Pastebin. By abusing these benign services, adversaries blend malicious traffic into normal network activity, dramatically reducing the effectiveness of conventional network-based defenses. When malware further encodes Command and Control (C&C) server addresses as hidden

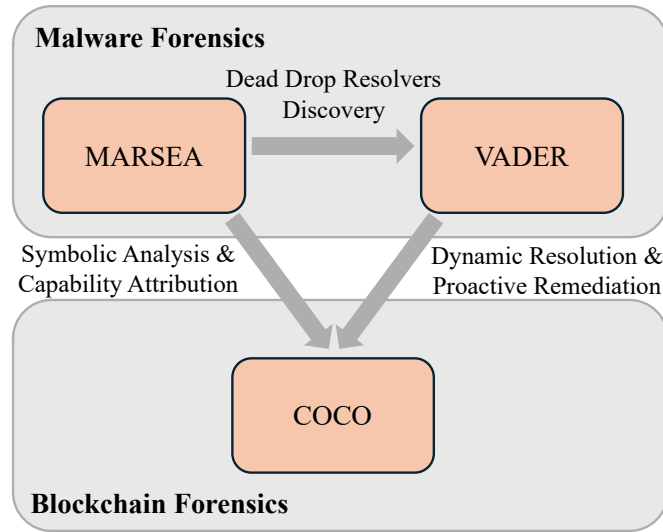


Figure 1.1: Interconnection of The Three Components of My Forensics Framework.

payloads within web application content (known as *dead drops*), even the removal of a single dead drop provides only temporary relief, as adversaries trivially redeploy new entries. Investigators lack automated tools that can map malware behavior to the full breadth of abused web services, recover all associated infrastructure, and enable proactive remediation at platform scale.

Blockchain-based financial fraud. The explosive growth of decentralized platforms, particularly Ethereum and its smart contract ecosystem, has enabled a new class of financially motivated cybercrime. Fraudulent smart contracts, orchestrated by anonymous actors using crypto wallets, have caused massive financial losses to victims worldwide. The blockchain’s pseudonymity and immutability render traditional fraud mitigation strategies inadequate: current responses are limited to flagging individual reported contracts as warnings, while the criminal entities behind them remain unconstrained and able to perpetually redeploy. No automated forensic pipeline existed to link individual fraud instances to the broader network of associated contracts, wallets, and campaign activities—the prerequisite for any proactive remediation strategy.

The following sections briefly introduce these techniques, outlining their specific technical contributions and the unique challenges they address.

1.2 Thesis Contributions

This thesis develops and validates the evidence-to-remediation paradigm through three systems—MARSEA, VADER, and COCO—that progressively deepen a single research agenda: automated evidence extraction followed by proactive, large-scale remediation. Each system targets a distinct facet of the reactive gap, and together they demonstrate that the paradigm generalizes across cybercrime domains.

- **MARSEA: Proactive Forensics of Web Application Abuse in Malware.** I present the first large-scale empirical study of malware that abuses web applications as command-and-control infrastructure. MARSEA uses concolic analysis to automatically identify which web applications a malware sample abuses and what vectors it exploits. It then employs session reconstruction and web application modeling to link extracted evidence to active malicious personas on those platforms. Deployed on 10,000 malware samples, MARSEA revealed 893 Web App-Engaged malware in 97 families abusing 29 web applications, uncovering a 226% increase in this category since 2020. Through collaboration with service providers, MARSEA enabled the takedown of 50% of identified malicious content.
- **VADER: Proactive Forensics of Dead Drop Resolver Malware.** MARSEA’s analysis exposed a particularly elusive subclass of web application abuse: Dead Drop Resolver (DDR) malware, which encodes C&C server addresses as manipulated payloads within benign web application content. Because these payloads are encoded or encrypted before being posted, the evidence extraction techniques in MARSEA alone cannot recover the full scope of the threat. VADER addresses this gap by localizing the DDR logic within malware binaries, isolating the de-manipulation boundary, and extracting the underlying *de-manipulation recipe* that governs how dead drops are encoded. Once extracted, these recipes enable web application providers to proactively scan their platforms for unknown dead drops—shifting remediation from reactive takedown to

forward-looking prevention. Applied to 100,000 malware samples, VADER identified 8,906 DDR malware samples, extracted seven distinct de-manipulation recipes, and uncovered 57.1% additional dead drops across 11 platforms. Collaboration with service providers achieved the removal of 94.4% of proactively discovered dead drops, disrupting over 6,674 active malware samples.

- **COCO: Proactive Forensics on Blockchain Fraud.** Extending the evidence-to-remediation paradigm beyond malware, I present the first automated forensic analysis pipeline for blockchain-based financial fraud that goes beyond individual contract flagging to trace and attribute entire fraud campaigns. COCO combines transaction-level forensics with symbolic analysis to recover all associated fraudulent contracts from a single fraud report, investigate recipient provenance, and attribute fraudulent capabilities to the Deceptive Creator Wallets (DCWs) orchestrating the campaigns. Applied to 157 confirmed fraud contracts, COCO uncovered 1,283,198 associated contracts linked to 91 DCWs, responsible for approximately \$2.09 billion in illicit profits traced back to September 2017. This work has led to direct collaboration with the U.S. FBI and Etherscan to pursue remediation.

Together, these three systems establish a generalizable framework for proactive cyber forensics: given a single incident report, automatically extract comprehensive forensic evidence and generate remediation strategies that neutralize threats at platform and ecosystem scale—shifting the forensic paradigm from reactive investigation to evidence-driven, large-scale intervention.

1.3 Thesis Overview

This thesis is organized as follows. Chapter 2 through Chapter 4 present the three projects. Chapter 5 presents conclusions and future research directions. Below, I provide a brief

overview of each project.

1.3.1 MARSEA: Hiding in Plain Sight: An Empirical Study of Web Application Abuse in Malware

Web applications provide a wide array of utilities that are abused by malware as a replacement for traditional attacker-controlled servers. Thwarting these Web App-Engaged malware requires rapid collaboration between incident responders and web app providers. Unfortunately, my research found that delays in this collaboration allow such malware to thrive. I developed MARSEA, an automated malware analysis pipeline that studies web app-engaged malware and enables rapid remediation. Given 10K malware samples, MARSEA revealed 893 malware in 97 families abusing 29 web apps. My research uncovered a 226% increase in the number of these malware samples since 2020 and found that malware authors are beginning to reduce their reliance on attacker-controlled servers—a 13.7% decrease in web app-engaged malware relying on attacker-controlled servers. To date, I have used MARSEA to collaborate with the web app providers to take down 50% of the malicious web app content.

1.3.2 VADER: Proactive Detection and Remediation of Dead Drop Resolver Malware

Dead Drop Resolver (DDR) malware evades traditional Command and Control server takedowns by dynamically resolving C&C addresses hosted on popular web applications, such as Dropbox and Pastebin. These addresses are often manipulated (encoded or encrypted), rendering existing detection techniques largely ineffective. To tackle this challenge, I introduce VADER, a malware forensics system specifically designed for the proactive detection of dead drops. Analyzing a dataset of 100K malware samples collected in the wild, VADER identified 8,906 DDR malware samples from 110 families that leverage 273 dead drops across seven web applications. Additionally, it proactively uncovered 57.1% more dead drops spanning 11 web applications. Case studies revealed that over 40% of

DDR malware samples employ sophisticated, layered de-manipulation algorithms. Beyond detection, VADER enabled proactive remediation by discovering 13 previously unknown dead drops from a single DDR malware sample, empowering web application providers to systematically scan their platforms.

1.3.3 COCO: Pulling the Strings: Proactive Forensics on Blockchain Fraud

Criminals, using crypto wallets referred to as Deceptive Creator Wallets (DCWs), have orchestrated fraudulent activities by luring victims to transfer funds to fraud smart contracts. Since it is almost impossible to reverse the transactions or pinpoint the true identity of the criminals, the industry has turned to flagging such contracts as user warnings. However, current mitigation efforts focus on individual contracts, overlooking the DCWs behind the scenes. Consequently, my research found that this oversight allows fraud to thrive. To address this, I developed COCO, an automated forensic analysis pipeline that processes a single fraud contract and generates evidence that the legal authorities need to mitigate the fraud. Applying COCO to 157 confirmed fraud contracts, my research uncovered 1,283,198 associated contracts linked to 91 DCWs, responsible for 2,638,752 ETH (\$2,089,504,682) in illicit profits. More alarmingly, COCO traces the fraudulent activities back to September 2017. In response, I am closely collaborating with Etherscan and the FBI to combat the fraud identified in my study.

CHAPTER 2

HIDING IN PLAIN SIGHT: AN EMPIRICAL STUDY OF WEB APPLICATION ABUSE IN MALWARE

Abstract

Web applications provide a wide array of utilities that are abused by malware as a replacement for traditional attacker-controlled servers. Thwarting these Web App-Engaged (WAE) malware requires rapid collaboration between incident responders and web app providers. Unfortunately, my research found that delays in this collaboration allow WAE malware to thrive. I developed Marsea, an automated malware analysis pipeline that studies WAE malware and enables rapid remediation. Given 10K malware samples, Marsea revealed 893 WAE malware in 97 families abusing 29 web apps. My research uncovered a 226% increase in the number of WAE malware since 2020 and that malware authors are beginning to reduce their reliance on attacker-controlled servers. In fact, I found a 13.7% decrease in WAE malware relying on attacker-controlled servers. To date, I have used Marsea to collaborate with the web app providers to take down 50% of the malicious web app content.

2.1 Introduction

Web applications provide a wide range of utilities, including content delivery [4], [5], [6], [7], [8], data storage [8], [9], [10], [11], social networking [12], [13], [14], and much more. These utilities incentivize malware authors to integrate popular web apps [15], [16], [17], [18] into their malware. This integration enables Web App-Engaged (WAE) malware to profile a victim's system, retrieve additional payloads, exfiltrate and store victim information, and even hide subsequent connections to attacker-controlled C&C servers. Even worse,

WAE malware are stealthier because their network traffic blends in with non-malicious traffic. Stopping this abuse requires technical and policy solutions from Incident Responders (IRs) and web app providers. However, such novel collaboration has not happened to date, and little research has been done to prove that WAE malware are prevalent enough to warrant such an investment.

Recent works on web app security show the possibility of abusing benign services for malicious purposes [19], [20], [21], [22], [23]. Yet, these works only scratch the surface of the WAE malware problem by solely considering proof-of-concepts [19], [20], [21] of possible abuse. The research community also proposed feature sets based on user engagement [22], [23], [24], [25], [26], [27] to classify abuse, where *engagement* is the interaction (e.g., views, likes, commits) that end-users have with web app content. Unfortunately, these ad-hoc studies focused on specific web apps and omitted the WAE malware required to attribute the abuse. Separately, prior research performed case studies of individual WAE malware samples [28], [29], [30] but only provided anecdotal evidence.

Aiming to give IRs and web app providers a clear view of WAE malware in the wild, this paper seeks to answer five research questions:

RQ1: How prevalent are WAE malware?

RQ2: What capabilities do web apps provide attackers?

RQ3: How effective is the current collaboration between IRs and web app providers?

RQ4: Can the abused web app help estimate the extent of a WAE malware’s infection?

RQ5: How should IRs and web app providers collaborate to effectively remediate WAE malware?

To answer my research questions, I studied WAE malware in collaboration with Netskope, the leading Secure Access Service Edge (SASE) provider, which provides cloud security and networking to more than 25% of the Fortune 100. I built Marsea,¹ an automated concolic analysis pipeline, to perform a scalable and retroactive study of WAE malware and answer

¹MalwAre Replace malicious Servers with wEb app Abuse.

my five **RQs**. Given only a malware binary, Marsea uses a novel *session reconstruction* and *web app modeling* approach to identify and *partition* the abuse vectors based on web app identities and assets. *Identities* are identifiable information (e.g., accounts, user names, API keys). *Assets* are content (e.g., files, posts) hosted on web apps. These partitioned vectors reflect the capabilities that the web app provides to the malware and serve as proof-of-abuse, enabling collaboration. Marsea then performs *engagement data harvesting* from the abused web apps, which gives IRs and web app providers an automated approach to measure the effectiveness of their collaboration. Further, I present three case studies to motivate the need for Marsea in their future novel collaboration.

To evaluate the prevalence of WAE malware, I deployed Marsea on 3K samples pulled randomly from VirusTotal [31] spanning the last 3 years. From this dataset, Marsea found that the number of web app abuse increased by 226% since 2020. Further, a more comprehensive study requires a larger set of known WAE malware. To this end, my collaborator gave me an additional 7K suspected WAE malware samples to augment my existing dataset. Marsea’s study of all 10K samples uncovered 893 malware in 97 malware families abusing 29 web apps. Worse still, Marsea revealed significant delays in the current collaboration between IRs and web app providers. I found an average delay of 253 days between when WAE malware authors create their assets on the web apps and when the WAE malware is finally detected. My findings also suggest that WAE malware could have infected up to 909,788 victims by evaluating the engagement data from 33 abused web app assets. At the time of writing, I have used Marsea to collaborate with the web app providers to take down 50% of active WAE assets, and I am awaiting responses for the remainder. Lastly, I have made Marsea available at: <https://github.com/CyFI-Lab-Public/MARSEA>.

2.2 Challenges and Motivation

RQ1: A temporal measurement of malware web app adoption is required to answer **RQ1**. However, many web app assets used by malware are no longer available. Thus, failing to

build this connection leaves subsequent network communications undiscoverable. Worse still, tracking only web app connections is not evidence of abuse (e.g., malware often connect to web apps to test network connectivity). Motivated by this, Marsea performs concolic analysis with a customized exploration strategy to spoof network communications. During my preliminary study, Marsea concolically analyzed the Zusy WAE malware and confirmed its connection to the Pastebin web app [8]. Then, Marsea identifies that the fetched content is written to a local file, confirming that Zusy is not using Pastebin for a connectivity check. In fact, Zusy hosts the configuration of the cryptocurrency miner XMRIG in the Pastebin post. Further, Marsea confirmed that Zusy does not rely on attacker-controlled C&C servers and only abuses Pastebin during its execution.

RQ2: Armed with Marsea, my study (§2.5) revealed that WAE malware are increasing in prevalence, which should alert the IRs and the web app providers to be prepared for future growth. Ideally, proof-of-abuse, including the web app capabilities being abused by each identity and asset, is needed to enable rapid remediation. Deriving this proof motivated my measurement of capabilities that web apps provide to malware **RQ2**.

Capability identification requires partitioning the WAE malware’s abuse vectors based on their identity and assets. To identify the abuse vectors, Marsea uses *Vector Rules* (§2.3.2) to match the information flow from the fetched content. By tracking the information flow from the content fetched by Zusy to the `WriteFile` operation, Marsea identified the File Download vector. Unfortunately, partitioning is challenging since the identity and asset are not readily available in the WAE malware binary. Investigating Pastebin abuse requires the Post’s content (the Pastebin *asset*) and the malware author’s Username (the Pastebin *identity*). Upon investigation, I found that Zusy’s binary includes no asset or identity but only a Pastebin paste key as the object name when configuring the network session with Pastebin. To overcome this challenge and obtain the identity and asset used by Zusy, Marsea performs session reconstruction (§2.3.3) to reproduce the Pastebin session (i.e., `GET pastebin.com/raw/QKAdpkY`). Then, Marsea uses Web App Modeling

(§2.3.4) to parse the session and extract the paste key `QKAdpkY`. Given the identified key and modeling, Marsea requests Pastebin and confirms that Zusy is abusing the file hosting capability of Pastebin via the asset (a post named `active`) from the identity `anthrax_`.

RQ3: Next, I turned my attention to studying how effective the current collaboration between IRs and web app providers is. To do so, I define two key time windows during a WAE malware outbreak. The *Response Delay (RD)* is the time between when the attackers create the assets to when the corresponding WAE malware (that abuse those assets) are first submitted to VirusTotal [31]. The *Post-Detection Window (PW)* begins when the WAE malware is first submitted to VirusTotal and ends when web app providers take down the associated assets. Minimizing *RD* and *PW* indicates a more effective collaboration. Upon closer investigation, I found that web apps track the time information of the hosted assets and often expose such data to the public. Motivated by this, Marsea queries the time information of assets from the web app and calculates the *RD* of Zusy as 21 days and *PW* as 108 days. Worse still, when I began this research, the post was still available on Pastebin. During my study, Marsea revealed a high *RD* and *PW*, leading to significantly more infected victims.

RQ4: Given that the current collaboration fails to quickly take down abused assets, I wondered how many victims end up infected by WAE malware. Several works have proposed covert access [32] or active monitoring [33], [34] of attacker-controlled servers. However, IRs still need to collect information manually to evaluate the extent of infections. This is hardly scalable and not applicable to WAE malware since the malware’s activity takes place on a public web app. In fact, the publicly available data gives IRs and web app providers a unique opportunity to monitor the extent of a WAE malware family’s infection by harvesting the web app’s engagement data (e.g., views, comments). Armed with this insight, Marsea harvested the view numbers of the Pastebin post abused by Zusy malware, which indicates up to 796 victims.

RQ5: Noting the deficiency of current collaboration efforts (**RQ3**) and the extent of WAE malware infections (**RQ4**), I develop my techniques in Marsea to target rapid remediation

of WAE malware. For Zusy, I submitted proof-of-abuse derived from Marsea to Pastebin, resulting in the revocation of abused identities and assets. Apart from Zusy, Marsea enabled me to correlate the assets attributed to WAE malware with other identities and assets on the web app to facilitate the remediation (designated *Lateral Remediation* in §2.5.5). Moreover, modern malware perform their infection via multiple stages, and WAE malware often fetch these stages from web apps. In fact, this research has found that this infection chain can reveal the connections among different WAE malware families and can assist IRs in uncovering new WAE malware from a single WAE sample (designated *Early Stage Termination* in §2.5.5). Lastly, Marsea also exposes an opportunity to monitor and counteract the migration of WAE malware intra- and inter-web apps (designated *Migration Remediation* in §2.5.5).

2.3 Methodology

Marsea needs only a malware sample binary as input. The analysis is automated to identify WAE malware and output proof-of-abuse and answers the 5 **RQs**. Marsea is open source and available at: <https://github.com/CyFI-Lab-Public/MARSEA>.

2.3.1 Concolic Exploration Strategy

Marsea uses concolic analysis to spoof network sessions to ensure forced execution of the malware. Specifically, Marsea introduces symbolic data through system *input* APIs that read data from external sources (e.g., network communication). The symbolic data enables Marsea to drive the execution when the assets are unavailable. As stated in §2.2, the Zusy malware abused Pastebin to host a malicious configuration file. During analysis, Zusy invokes `InternetReadFile` to fetch the content from Pastebin. Marsea introduces symbolic data to the buffer defined by `InternetReadFile`. When Zusy checks the string (e.g., `cpu`) in the buffer, Marsea can generate two states to ensure continued execution even if the abused asset is unavailable.

However, since I/O APIs are expected to be frequently invoked by the WAE malware, and Marsea injects symbolic data for each, numerous states are generated. For example, when Zusy fetches content from Pastebin, it continues to invoke `InternetReadFile` in a loop until all data has been read, which would be reflected as the parameter `lpdwNumberOfBytesRead` being zero. In this case, Marsea injects symbolic data λ to `lpdwNumberOfBytesRead`. Now, every time the Zusy checks if λ_i is zero in the i -th interaction, Marsea spawns two states with $\lambda_i > 0$ to keep the execution in the loop and with $\lambda_i \equiv 0$ to drive the execution out of the loop. However, if Marsea chooses the state randomly, it could easily re-visit the loop. To solve this, I developed a context-aware exploration strategy.

Specifically, Marsea checks if the control flow being executed: (1) is from the malware or a Dynamic Linked Library (DLL) and (2) if it has already been analyzed or executed before. Based on these, each available path will be in any of four possible conditions:

$$\begin{aligned} (\text{new code, malware}) &= \alpha & (\text{old code, malware}) &= \gamma \\ (\text{new code, library}) &= \beta & (\text{old code, library}) &= \delta \end{aligned}$$

Marsea prioritizes each state to explore by following an order of precedence: $\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta$. Now, Marsea will prioritize the state leaving the iteration immediately ($\lambda_0 = 0$) and the state executing the iteration once ($\lambda_0 \neq 0 \wedge \lambda_1 = 0$). This context-aware exploration strategy enables Marsea to efficiently handle loops and innumerable states (e.g., read until nothing).

2.3.2 Web App and Vectors Identification

Concolic analysis proves efficient for selective path exploration, but, on its own, it cannot answer my 5 **RQs**. To answer **RQ1**, Marsea needs to identify abused web apps and the abuse vectors performed by the malware. This is similar to a detective (e.g., IRs) collecting evidence from a crime scene (e.g., malware) for off-site processing. Without the context

Algorithm 1: Marsea’s Vector Identification.

Input: Instruction I , Vector Rule R , Connect API Annotation Π , Current State S

Output: Identified Vectors with associated web app Vec

```
// Initialize vector and propagation graph
1  $Vec \leftarrow \emptyset$ ;
2  $Prop \leftarrow \emptyset$ ;
// Invoke connect function
3 if  $I.target \in \Pi$  then
    | // Update handle to web app map
4    |  $S.M[DefHandle(I)] \leftarrow GetTarget(I)$ ;
5 end
// I uses handle
6 if  $(UseHandle \leftarrow I.ARGs \cup S.M) \neq \emptyset$  then
7    |  $S.M[DefHandle(I)] \leftarrow S.M[UseHandle]$ ;
8 end
9 for  $R_{ij} \in R$  do
    | // Invoked function is in rule  $R_{ij}$ 
10    if  $I.target \in R_{ij}$  then
        | // Initialize instruction list to trace back
11        |  $T \leftarrow [I]$ ;
12        while  $Inst \leftarrow Pop(T)$  do
            | // Add inst in graph Prop
13            |  $Prop.addNode(inst)$ 
            | // inst uses handle in S.M
14            | if  $inst.ARGs \cup S.M \neq \emptyset$  then
15                |  $Vec_{ca} \leftarrow S.M[inst.ARGs \cup S.M]$ ;
16            | end
            | // Trace the symbolic data
17            | for  $NewInst \in SymbolicSource(I.ARGs)$  do
                | // Build the dependence
18                |  $Prop.addEdge(NewInst, Inst)$   $T.Append(NewInst)$ ;
19            | end
20        | end
        | // Try matching Prop with  $R_{ij}$ 
21        | if  $HasPath(Prop, R_{ij})$  then
22            |  $Vec_{behavior} \leftarrow R_{ij}$ ;
23            | break;
24        | end
25    | end
26 end
27 return  $Vec$ ;
```

of the crime scene, the detective may make (inaccurate) assumptions about evidence. To overcome this challenge, Marsea uses a fined-grained and context-driven approach to identify abused web apps and abuse vectors.

Algorithm Algorithm 1 shows how Marsea identifies abused web apps and vectors. During concolic analysis, Line Line 3 of Algorithm Algorithm 1 tracks the invocation of

system APIs which build the connection to remote endpoints (e.g., `WinHttpConnect`), denoted as Π . Once the malware invokes a system API in Π , Marsea recovers the communication target information from the input to the API (*GetTarget()* of Algorithm Algorithm 1). For example, when Zusy builds the network connection to the remote endpoint by invoking `InternetConnect`, Marsea identifies the endpoint `pastebin.com` from parameter `lpszServerName`. Notably, since Marsea’s concolic analysis logic needs no information of Π annotation, it is a one-time effort and can be extended easily. Armed with the ability to identify abused web apps, Marsea can pursue large-scale analysis to identify the prevalence of WAE malware (**RQ1**).

Table 2.1: Abuse Vectors And Associated Rules.

Definitions for Abuse Vector Rules		
Primitive	Example Library Functions	
<i>Receive</i> :	<code>WinHttpReadData WinHttpReceiveReponse ...</code>	
<i>ReadFile</i> :	<code>ReadFile ReadFileEx ReadFileScatter ...</code>	
<i>SysOp</i> :	<code>RegOpenKey CreateFile DeleteFile ...</code>	
<i>WriteFile</i> :	<code>WriteFile fwrite write ...</code>	
<i>Send</i> :	<code>send HttpSendRequest WinHttpSendRequest ...</code>	
<i>Connect</i> :	<code>WinHttpConnect InternetConnect connect ...</code>	
<i>Execute</i> :	<code>ShellExecute CreateProcess ...</code>	
Abuse Vectors	Priority	Rules
Execute	R_{11}	$(Receive \rightarrow WriteFile) \rightarrow Execute$
File Download	R_{12}	$Receive \rightarrow WriteFile$
Message Fetch	R_{13}	$Receive \rightarrow SysOp$
File Upload	R_{21}	$ReadFile \rightarrow Send$
Message Push	R_{22}	$SysOp \rightarrow Send$
Jump Server	R_{31}	$Connect \rightarrow Receive \rightarrow Connect'$

Answering **RQ2** requires Marsea to identify the abuse vectors, which indicates the capabilities the web apps provide. To do so, Marsea tracks information flow in the context of malware execution. Unfortunately, tracking all concrete data during analysis is unnecessary and unscalable. Recall the symbolic data Marsea introduced to enable the concolic analysis (§2.3.1). It turns out that the propagation of symbolic data indicates the information flow. As shown in Lines Line 10 to Line 13 of Algorithm Algorithm 1, Marsea tracks symbolic data propagation, saved in *Prop*. For example, after Marsea introduces symbolic data through

`InternetReadFile` invoked by Zusy, the symbolic data is propagated to `WriteFile`. In this case, Zusy is abusing the web app to perform the *File Download* vector. These API dependencies Marsea uses (e.g., `InternetReadFile` $\rightarrow$ `WriteFile`) are presented as vector rules, R_{ij} in Table 2.1. The full list of system APIs will be open-sourced along with Marsea. I present the definition of each abuse vector below:

Execute: The malware retrieves an asset from a web app, stores the content in a local file, and executes the file on the victim’s system.

File Download: The malware retrieves an asset from a web app and stores the content in a local file on the victim’s system. These are often configuration files for the malware.

Message Fetch: The malware retrieves an asset from the web app for malware’s internal use (e.g., C&C commands). Specifically, Marsea tracks that the retrieved asset data is stored in the malware’s memory and used in subsequent operations (e.g., compared to a hard-coded string or used as a decryption key) but never written to a local file on the victim’s system.

File Upload: The malware reads one or more files on the victim’s system and uploads the data to an asset on the web app.

Message Push: The malware sends data to an asset on the web app. The data does not originate from a file on the victim’s system. This is commonly used to exfiltrate victim information (e.g., IP, username).

Jump Server: The malware retrieves an asset from a web app which is used to resolve a new connection endpoint.

As shown in Table 2.1, I define the rule primitives based on the vectors, which represent a set of APIs that perform a task that the primitive is named for. Column 1 (bottom half of the table) lists the abuse vectors considered in this study. To decide on the abuse vector rules, I referenced reports from security experts [35], [36] and the MITRE ATT&CK Framework [37] to understand how malware vectors are generally applied. Column 2

presents the matching priority for each vector. Specifically, $R_{i,j}$ has a higher priority than $R_{i,m}$ when $j < m$. Given $Prop$ and the Rule set, Marsea identifies abuse vectors by evaluating if the path presented by R_{ij} exists in $Prop$ (Line Line 21 of Algorithm Algorithm 1). Since the exploration logic does not depend on vector identification, vector rules are both *composable* and *extensible* with no need to re-analyze.

Up until now, Marsea has identified Zusy contacting Pastebin by invoking `InternetConnect` and performing the File Download vector by verifying the information flow from `InternetReadFile` to `WriteFile`. However, to conclude that Zusy is abusing Pastebin, Marsea needs to find the connection between `InternetConnect` and `InternetReadFile`. To do so, Marsea reveals the connection using *handles* (e.g., `HINTERNET`, `socket`), shown in Lines Line 3 to Line 7 of Algorithm Algorithm 1. For each exploration state S , Marsea maintains a map M , which maps the handles defined by system APIs to the web app. When Marsea identifies vectors, it checks the map to attribute vectors to abused web apps. For example, amid Zusy’s analysis, Marsea tracks the handle through `InternetConnect` $\rightarrow$ `InternetOpenRequest` $\rightarrow$ `InternetReadFile`. Armed with the ability to attribute the abused web app to the malicious vector (e.g., File Download), Marsea confirms Zusy abusing the file hosting capability of Pastebin (**RQ2**).

2.3.3 Session Reconstruction

Unfortunately, reporting Zusy and the abuse vector to Pastebin is not enough to pinpoint the abuse on the platform and remediate the abuse. Instead, proof-of-abuse, including partitioned abuse vectors based on identities and assets, is needed. Since WAE malware abuse web apps through network sessions, identifying the identities and assets requires Marsea to extract the network sessions first.

However, given the multi-path nature of concolic analysis, acquiring in-order network sessions at the operating system layer is challenging. For example, concolic analysis can

switch the exploration state after Zusy invokes `InternetConnect`. Then, on the system level, the new network sessions appear to be appended after `InternetConnect` of the old state, which is not true in the malware. To overcome this challenge, I introduce a graph automata model of network session states, which enables Marsea to reconstruct sessions by interleaving different exploration states. As shown in Figure 2.1, I generalized the session states into opened ($\textcircled{o}$), connected ($\textcircled{c}$), sent ($\textcircled{s}$), and closed ($\textcircled{cl}$). The transitions between different states fall into five categories: (1) a transition from the opened state to a connected state by connecting to the target (function $h_o.c()$), (2) a transition from the old connected state to a new connected state by assigning session parameter information through headers (function $h_c.a()$) and parameters (function $h_c.p()$), (3) a transition from the connected state to sent state by sending the request (function $h_c.s()$), (4) a transition to the closed state by freeing the resource (function $h_s.cl()$ and $h_o.cl()$). Each transition function in Figure 2.1 presents a group of system APIs. During the analysis of Zusy, the session state transition $\textcircled{o} \rightarrow \textcircled{c}$ happens when Zusy invokes `InternetConnect`. As stated in §2.3.2, Marsea extracts the abused web app, e.g., `pastebin.com`, during this transition. The next session state transition Marsea observed in Zusy was $\textcircled{c}$ to $\textcircled{c}$, caused by invoking `InternetOpenRequest`. During this transition, Marsea extracts the session parameters. For Zusy, this is the verb (GET) and object name (pQKAdpkY). When Zusy invokes `HttpSendRequest`, the session state transitions from $\textcircled{c}$ to $\textcircled{s}$. Note the transition from $\textcircled{s}$ to $\textcircled{cl}$ can include more operations (e.g., receiving). I abstracted those into a dotted line since Marsea only needs sessions composed by the malware to extract the abused identities and assets. Given Marsea’s session graph automata model, it reconstructs multiple network sessions by interleaving execution states.

2.3.4 Web App Modeling

Given reconstructed network sessions, Marsea is ready to identify the identities and assets. However, since web apps do not share the same session syntax, it is challenging to parse

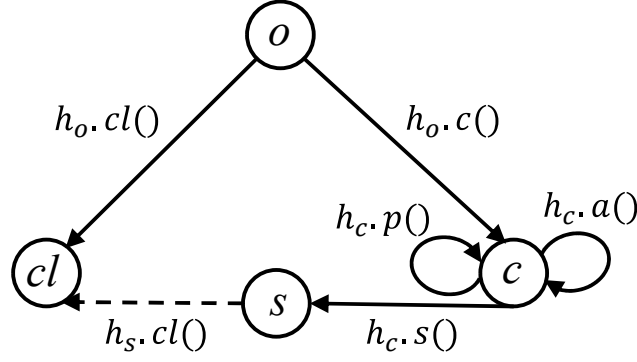


Figure 2.1: Graph Automata Model To Guide Session Reconstruction From Interleaving Execution States.

the reconstructed network sessions. Luckily, each web app provides public documents to regulate the network sessions. Motivated by this, I proposed a web app model, $\Phi = (\phi_\alpha, \phi_\beta)$, to model the API described in the web app’s documentation. ϕ_α includes the compositions of the public frontends of the web apps. $\phi_\beta = [\theta_1, \theta_2, \dots]$ includes web app APIs. Each web app API θ_i embeds the response format to annotate the identity, asset, time, engagement, and raw data. Now, Marsea can uncover the web app’s identifier specific to the abused asset from the reconstructed network sessions. Then, Marsea uses the identifier and web app APIs, ϕ_β , to harvest and parse the information fetched from the web app. For example, to query the abused post from Pastebin through the public frontend, Zusy appends a Pastebin paste key QKAdpkY after the URL `pastebin.com/raw`. Using ϕ_α , Marsea extracts the paste key from the reconstructed network sessions. Meanwhile, Pastebin APIs `api_scrape_item` and `api_scrape_item_meta` can be used to query the raw data and the metadata information, respectively. By modeling these APIs, Marsea queries Pastebin using the paste key and parses the response to uncover: (1) the raw data of the abused post, (2) the identity (Username `anthrax_`), (3) asset (post named `active`), (4) creation time (12-19-2021), and (5) the engagement data (796 views).

With the identity, Marsea uses the modeled frontend (ϕ_α) and modeled APIs (ϕ_β) to query the frontend and APIs accepting the identity as input. For Zusy, Marsea uses modeled frontend `pastebin.com/u/` to query other posts from `anthrax_`. Notably, given the

finite number of web apps and the APIs they provided, Marsea can be easily extended to support additional web apps. Till now, the partitioned vectors constitute the admissible proof-of-abuse, which enables fast collaboration. (**RQ5**). Meanwhile, Marsea can evaluate the extent of infection using the harvested engagement data (**RQ4**). By querying the first detection date of the malware and comparing the date with the harvested time information, Marsea can calculate the Response Delay and the Post-detection Window (**RQ3**).

2.4 Validating My Techniques

Implementation. Marsea is implemented in C++ (12K lines) leveraging S2E [38], which has been used in top-tier research, for concolic analysis [39], [40], [41], and Python (8K lines) to attribute identified malicious vectors to abused identities and assets and harvest corresponding engagement information on the abused web app platforms. I used AVClass2 [42], a state-of-the-art malware labeling tool, to label my dataset. I used an Ubuntu 20.04 LTS system to host Marsea. Given one malware, Marsea spawns a Windows 7 virtual machine (20GB RAM and 6 x 3.9GHz CPUs) based on the snapshot of a bare-metal machine to mimic the victim system.

In this section, I validate Marsea’s ability in (1) abuse vector identification, (2) attributing vectors to abused identities and assets, and (3) comparing the identified vectors with concrete execution. I randomly selected malware samples from my dataset until I found those with industry security reports, resulting in 20 WAE malware from 11 families. I manually reverse-engineered these samples and used industry reports to derive ground truth².

2.4.1 Vector Identification

Table 2.2 presents Marsea’s abuse vector identification evaluation. Columns 1-4 list the abused web apps, malicious vectors, malware families, and the number of samples (#V) in my ground truth dataset. Column 5 presents the ground truth results (G) derived from

²Note that Marsea does not need nor have access to this ground truth data.

Table 2.2: Evaluation Of Abuse Vector Identification And Identity and Assets Attribution.

Web Apps	Vec	Family	#V	G	C	M	T	P	F	P	F	N	#ID	#A	TPFN	Path	
													G	M			G
GitHub	EXE	Ramnit	1	2	0	2	2	0	0	0	1	1	2	2	3	0	887
		Apost	2	2	0	0	0	0	2	1	0	1	0	0	0	2	1,117
		Zorex	5	5	0	5	5	0	0	0	1	1	1	1	2	0	943
		Dldr	1	1	1	1	1	0	0	1	1	1	1	2	0	1,043	
	FD	Apost	2	4	0	6	4	2	0	1	1	2	2	3	0	1,117	
		Dldr	1	1	1	1	1	0	0	1	1	1	1	2	0	1,043	
Twitter	JS	Vtflooder	5	5	0	5	5	0	0	1	1	1	1	2	0	279	
Dropbox	EXE	Banload	1	1	0	1	1	0	0	1	1	1	1	2	0	1,503	
		Lowball	1	2	0	2	2	0	0	1	1	2	2	3	0	2,474	
	FD	Lowball	1	4	0	4	4	0	0	1	1	4	4	5	0	2,474	
	FU	Lowball	1	2	0	2	2	0	0	1	1	2	2	3	0	2,474	
	MP	Lowball	1	3	0	3	3	0	0	1	1	3	3	4	0	2,474	
VrsTotal ³	FU	Vtflooder	5	5	0	5	5	0	0	1	1	1	1	2	0	279	
GDrive	EXE	Ramnit	1	1	1	1	1	0	0	1	1	1	1	2	0	803	
		Formbook	1	1	1	1	1	0	0	1	1	1	1	2	0	341	
Telegram	MF	Sstealer ¹	1	1	1	1	1	0	0	1	0	1	1	1	1	1,631	
Pastebin	JS	Msil	2	2	0	1	1	0	1	1	1	1	1	2	0	803	
	MF	Dkomet ²	1	1	0	1	1	0	0	1	1	1	1	2	0	1,325	
Total	All	11	20	43	5	42	40	2	3	18	16	27	26	42	3	1,052 ⁴	

1,2,3: Short for Stellarstealer, Darkkomet and VirusTotal

4: Average explored paths for each sample

#ID: Number of identities #A: Number of assets

the industry report and manual investigation. Column 6 presents the concrete execution results (C). Specifically, I count the vectors the concrete execution *could* reveal by manually verifying the information flow. The results generated by Marsea (M) and the evaluation metrics are presented in Columns 7-10, including True Positive (TP), False Positive (FP), and False Negative (FN). The paths Marsea explored are shown in the last column.

Overall, Marsea correctly (TP) identified 40 abuse vectors across 20 malware samples. As shown in Column 5 of Table 2.2, compared with single-path concrete execution, Marsea identified 37 more vectors, accounting for 86% of the vectors in the ground truth. Understandably, since remote endpoint network communication can be implemented through multiple API calls (e.g., `socket`, `connect`, `recv`), single-path exploration can halt during connection establishment if the endpoint is unavailable. Next, my manual investigation found 43 abuse vectors, revealing an overall accuracy of 93%. As seen in Row

2, Row 5, and Row 17 of Table 2.2, Marsea produced 2 FPs and 3 FNs when it analyzed Apost and Msil. For each variant of Apost, my manual investigation revealed that the malware retrieved 3 files from a GitHub repository. One of these files is a binary executed by the malware, which accounts for the 1 Execute (EXE) abuse vector, while the other 2 files were dropped as configuration files. However, Marsea reported no EXE vector (Row 2) and 3 File Download (FD) vectors (Row 5). Similarly, for one Msil variant, my manual investigation uncovered Pastebin abuse to resolve the new network connection target. However, Marsea reported no Jump Server (JS) vector. Upon closer inspection, I found that both cases were due to unresolved symbolic constraints accumulated through exploration, which further impedes Marsea’s exploration. I confirmed these are rare occurrences. Note, since the FD vector is the precondition of the EXE vector (the malware needs to download the file before executing it), in the case of Apost, the FP is the direct cause of the FN. Therefore, once Marsea can escalate FD vector to EXE vector, it will not introduce any FPs or FNs. Overall, Marsea was 93% accurate in abuse vector identification, making it robust for my large-scale study.

2.4.2 Identities & Assets Attribution

While analyzing the ground truth dataset, Marsea located the identities and assets during exploration. I compared the results Marsea generated with my manually-derived ground truth and concrete execution, and the results are presented in Columns 10-15 of Table 2.2. Specifically, Columns 11-16 show unique identities (#ID) and assets (#A) Marsea identified. Columns 17-19 present Marsea’s evaluation metrics, including the True Positive (TP), False Negative (FN), and explored paths. To illustrate, Row 9 shows the results of Marsea’s analysis of Lowball. Marsea revealed 2 EXE vectors tied back to 2 assets (files) hosted on Dropbox using 1 identity (API key). Overall, Marsea found a total of 42 different assets and identities out of 45 manually-derived assets and identities in 20 WAE malware samples, which yield over 93% accuracy.

Next, Marsea introduced 3 FNs when analyzing malware Apost and Stellarstealer. The

FN in Apost is caused by the prior vector identification FN in §2.4.1 (if Marsea misses a vector, then it can not attribute that vector). Upon closer inspection, I found that the FNs in Stellarstealer are because the abused Telegram channel is no longer available. Even though Marsea can extract the asset’s content (Telegram channel) from the network session, Marsea cannot identify the identity (i.e., Username) since that information is hosted in the channel and not included in the network sessions. Still, given the low number of FNs and over 93% accuracy, Marsea provides the means to attribute abuse vectors to identities and assets effectively.

2.5 WAE Malware Study Insights

To study the prevalence of WAE malware (**RQ1**), I deployed Marsea on 3K malware randomly pulled from VirusTotal [31], spanning 2020-2022 (1K samples per year). Specifically, I used VirusTotal Intelligence Search [43] to fetch signatures of all Windows PE malware with a first submission date in each of the 3 years and at least 5 AV engine detections, indicating maliciousness [44]. Then, I randomly selected 1K signatures from each year and deployed Marsea on the fetched samples. For **RQs 2-5**, I extend my dataset with an additional 7K suspected WAE malware samples collected by Netskope (totaling 10K samples) to investigate WAE malware-specific topics.

2.5.1 WAE Malware Prevalence

Marsea extracted communication endpoints from the 3K samples randomly pulled from VirusTotal. Based on the endpoints, the malware are categorized as WAE-only (i.e., communicates solely with web apps), hybrid (i.e., communicates with both web apps and attacker-controlled C&C servers), and malicious backend-only (i.e., communicates solely with attacker-controlled C&C servers). Marsea evaluates the reputation of the endpoint effective Second-Level Domain (eSLD) using Tranco 1M top sites [45] and Cisco Talos [46]. If the eSLD is categorized as *untrusted* by Cisco Talos or is not listed in the

Table 2.3: RQ1 Malware Reliance On WAE And Year Over Year (YoY) Increase Of WAE Adoption.

Year	Vector	WAE	Hybrid	Malicious	Total
2022	Execute	47	3	177	227
	File Download	38	6	143	187
	Message Fetch	10	0	255	265
	Jump Server	33	0	21	54
	File Upload	15	3	274	292
	Message Push	41	2	235	278
	Subtotal +22% YoY	173	14	813	1,000
2021	Execute	52	0	218	270
	File Download	34	9	147	190
	Message Fetch	5	0	213	218
	Jump Server	19	0	33	52
	File Upload	9	7	297	313
	Message Push	31	0	144	175
	Subtotal +166% YoY	141	16	843	1,000
2020	Execute	21	1	329	351
	File Download	13	3	158	174
	Message Fetch	7	0	293	300
	Jump Server	4	1	79	84
	File Upload	3	0	194	197
	Message Push	5	0	53	58
	Subtotal	53	5	942	1,000
Total +226% 2020→2022		367	35	2,598	3,000

Tranco top sites, Marsea considers it an attacker-controlled C&C server.

Notably, malware can contact endpoints only to verify the network connectivity of the infected victim system. Failing to exclude these cases would lead to a biased prevalence measurement. However, Marsea would not report any abuse vector for a connectivity check since the information flow (e.g., status code) fetched from the websites would not flow to the system APIs Marsea modeled. The overall results are presented in Table 2.3. I made several observations from Table 2.3: (1) As shown in the Total row of Table 2.3, in the last three years, malicious backend-only malware, WAE-only malware, and hybrid malware account for 2598 (86.6%), 367 (12.2%), and 35 (1.2%) of all 3K malware. (2) As shown in Column 5 of Table 2.3, the malware using only malicious backends has decreased by 13.7%. (3) Marsea found notably few hybrid malware (only 35 across all 3 years), which suggests that WAE malware authors prefer to shift entirely to web apps. (4) From the

Total row of Table 2.3, the WAE-only malware has increased by 226% in the last three years (increased by 22% and 166% in 2022 and 2021), indicating the recent prevalence of WAE-only malware.

RQ1: Marsea identified 2,598 (86.6%) malware that rely on attacker-controlled servers. Marsea also revealed a **226%** increase in WAE-only malware since 2020, showing malware’s growing adoption of web app abuse. This steep increase should alert the IRs and the web app providers to be prepared for future growth.

2.5.2 WAE Malware Capabilities

Deploying Marsea on 10K malware samples confirmed 893 WAE malware across 97 malware families, abusing 29 web apps. Since WAE malware abuse vectors indicate the web app capabilities abused by the malware, I present the vectors Marsea identified in Table 2.4. I aimed to study the temporal adoption of each abuse vector on different web apps. To do so, I computed the First Abuse Date (*FAD*) and Last Abuse Date (*LAD*) for each vector v on every web app w , shown in Columns 6-7. The *FAD* is the earliest VirusTotal “first submission date” across all WAE malware samples found abusing v on w . The *LAD* is the most recent VirusTotal “first submission date” across all WAE malware samples abusing v on w (i.e., when VirusTotal discovered the most recent variant). Then, I measured the time difference to evaluate the abuse history of malware abusing v on w , shown in the History column of Table 2.4. Notably, the lifetime of abused assets can be longer than the abuse history, which will be evaluated in §2.5.3.

As shown in the Summary row of Table 2.4, Marsea found a total of 893 (8.9%) WAE malware in 10K samples. Notably, 32% of the 7K samples that Netskope suspected as being WAE malware were only performing network connectivity checks using web apps. As mentioned in §2.5.1, connectivity checks present no malicious vectors and are omitted from my study. This accounts for the drop in the percentage of WAE malware compared

Table 2.4: WAE Vector Landscape.

Category	Web Apps	Vector	Mal	Fam	<i>FAD</i>	<i>LAD</i>	History (Days)
File Sharing	GDrive	EXE	263	24	2020-01	2022-06	894
		FD	36	6	2020-03	2022-03	716
		JS	23	5	2019-06	2022-03	1,002
	Dropbox	EXE	13	6	2010-07	2020-10	3,734
		FD	1	1	2016-05	2022-02	2,091
		FU	4	3	2016-09	2021-07	1,756
	Pastebin	FD	5	3	2021-12	2022-04	130
		MF	26	11	2018-02	2022-02	1,463
		JS	25	10	2014-02	2022-04	2,990
	Zippyshare	EXE	1	1	2018-06	2018-06	3
Software Hosting	Imgur	EXE	1	1	2013-10	2013-10	1
	Tietuku	EXE	17	1	2016-05	2020-06	1,490
		FD	17	1	2016-05	2020-06	1,490
	GitHub	EXE	29	10	2020-09	2022-10	772
		FD	4	2	2020-09	2022-10	762
		MF	9	5	2019-10	2022-04	922
		JS	4	1	2020-03	2022-03	732
	SourceForge	EXE	9	1	2022-01	2022-02	33
		JS	11	6	2022-01	2022-02	20
Social Networking	Telegram	MF	4	2	2022-02	2022-03	30
	Twitter	JS	54	7	2014-06	2022-01	2,773
	Discord	EXE	49	11	2021-03	2022-01	314
		MP	37	9	2021-10	2022-03	153
	Slack	MP	13	4	2020-08	2022-03	585
	Facebook	MF	1	1	2018-07	2018-08	29
	V Kontakte	FD	1	1	2018-02	2020-01	705
Website Builder	Wordpress	EXE	3	2	2011-11	2014-05	914
		JS	9	3	2011-05	2020-05	4,026
	Webs	EXE	5	2	2010-03	2019-10	3,596
		FD	3	1	2012-05	2017-11	2,021
Blog	Tripod	EXE	1	1	2007-03	2014-05	2,628
		FD	1	1	2012-01	2021-07	3,474
	Blogspot	EXE	1	1	2018-07	2018-08	853
		FD	1	1	2011-11	2014-05	932
		MF	1	1	2018-07	2018-08	29
Dynamic DNS	Afraid	JS	110	8	2020-09	2022-03	544
	DuckDNS	JS	8	1	2022-02	2022-02	1
Local Info	Cmyip	MF	6	2	2011-11	2014-05	910
	Wikidates	MF	5	1	2022-03	2022-03	1
	Ip-api	MF	3	2	2020-10	2020-12	59
	wimip	MF	2	2	2011-11	2014-05	898
	Ip2location	MF	2	2	2011-11	2014-05	898
	Ip-address	MF	2	2	2011-11	2014-05	898
	Ipify	MF	11	6	2022-01	2022-02	27
	Timeanddate	MF	1	1	2022-03	2022-03	1
Anti-Virus	VirusTotal	FU	106	10	2014-06	2022-05	2,281
Summary	29	6	893 ¹	97 ¹	2007-03	2022-03	5,493

with §2.5.1.

Row 1 Columns 4-5 of Table 2.4 show that 263 (29.4%) WAE malware in 24 (24.7%)

families abuse GDrive to perform the Execute (EXE) vector. GDrive is the most abused web app, illustrating the attackers' preference for reputable web apps. Surprisingly, GDrive not only gives the WAE malware the capability of hosting malicious executables, as shown in Row 1, but also enables the resolution of next-hop endpoints, shown as JS vector in Row 3. Similarly, from Rows 37-38 (Dynamic DNS), Afraid and DuckDNS provide 118 WAE malware with the capability to resolve additional endpoints, protecting attacker-controlled C&C servers. Consequently, when the resolved endpoints are denylisted, the attackers can update the record in the web app to re-activate the WAE malware on the victim system.

From the category Local Info, Marsea found 8 web apps abused to query the IP, geo-location, and the local time of the victim system. Since identity and assets are not required to use these services, it is challenging to remediate the abuse. I suggest that these web app providers require an identity in the network session when providing service to prevent abuse by blocking the malicious identity. The wide range of capabilities and the benefit of the benign traffic incentivizes malware authors to integrate web apps, which explains the growth of WAE malware found in §2.5.1. Unnervingly, I found that this incentivization happened earlier than expected. As shown in the Summary row of Table 2.4, the WAE malware can be traced back to 2007, and the average abuse history of web apps can reach up to 5,493 days.

RQ2: Marsea revealed 893 WAE malware (97 families) abusing 29 web apps. Identifying 6 malicious vectors in WAE malware indicates a wide range of web-app-provided capabilities. Worse still, WAE malware in my dataset spans back to 2007, with an average of 5,493 days of abuse history for each abused web app.

2.5.3 Collaboration Evaluation

To evaluate collaboration effectiveness, I measure RD by calculating the time difference between the First Creation Date (FCD) of assets the WAE malware abused and the First

Table 2.5: Collaboration Evaluation By Measuring RD And PW of Live Identity (ID) and Asset (A).

Web Apps	Live Mal	#ID	#A	Live Assets Growth (2011-2022)	Response Delay				μ_{pw} (Days)
					Min	Max	Med	Avg	
GDrive	214	41	42		2	104	72	59	-59
GitHub	34	19	27		0	1,320	446	583	173
Pastebin	33	12	13		1	242	122	118	-118
Telegram	2	2	2		296	359	327	327	-328
Twitter	26	1	1		-51	-51	-51	-51	51
WordPress	3	3	3		5	38	37	29	1,635
Discord	21	21	21		7	100	34	51	-17
Blogspot	3	4	4		-72	12	7	-13	26
Dropbox	10	10	10		1	1,440	741	813	-813
Webs	1	1	1		398	398	398	398	-398
Afraid	110	1	1		1,812	1,812	1,812	1,812	-1,812
Tripod	1	1	1		-19	-19	-19	-19	19
Facebook	1	1	1		85	85	85	85	10
VK	1	1	1		1	1	1	1	1
DuckDNS	1	1	1		759	759	759	759	-759
Summary	430	119	129		0	1812	344	253	-49

Submission Date (FSD) of the WAE malware on VirusTotal. RD also presents the time window that the WAE malware *could* have to infect systems without being identified as malware. Similarly, I calculate PW using the time difference between the malware's First Submission Date (FSD) and the attacker's Last Update Date (LUD) of the assets the malware abused. Notably, Marsea uses web app modeling (§2.3.4) to harvest the information. Since the web app abuse identification does not rely on web app modeling, I curated the web app modeling based on the identified web apps during my study. The web apps and their available information are listed in §A.1.

The results of this evaluation are shown in Table 2.5. Note that WAE malware can abuse legitimate web app assets as its components (e.g., pulling legitimate proxy software), in which case they are omitted in Table 2.5 since RD and PW could not reflect the efficiency

of the current collaboration. I will further discuss how I identified the legitimate assets in §2.5.4.

Shown in Column 2 of Table 2.5, before I started collaborating with the web app providers, 96% WAE malware Marsea analyzed still had live assets on the abused web app platform. Column 5 shows that assets on Website Builders (i.e., WordPress, Tripod, Webs) are particularly long-lived. For example, Marsea identified the Symmi malware abusing the WordPress website `turrona5.wordpress.com` to resolve the next-hop endpoint. Surprisingly, the content on the website was posted in 2011 and is still live until 2022. Since I took down these assets by collaborating with web app providers (discussed in §2.5.5), it is evident that IRs failed to initialize the collaboration. Worse still, the Summary row of Table 2.5 shows that the *RD* of IRs reaches 253 days on average. In other words, analyzed WAE malware could go undetected for approximately 253 days. More interestingly, Column 10 shows the average *PW* (μ_{pw}) for each abused asset. A positive μ_{pw} (e.g., Github) indicates that the abused assets were updated even after the malware was detected.

RQ3: Marsea found that 48% of 893 WAE malware remained active until my study. An average RD of 253 days reveals deficient collaboration between the IRs and the web app providers. Marsea found some abused assets updated for hundreds or thousands of days after the malware was detected.

2.5.4 WAE Malware Engagement Data

In this section, I perform a large case study to estimate the extent of a WAE malware infection by analyzing the assets' engagement data. Starting with the 15 web apps shown in Table 2.5, 6 of them (which host 93 live assets) expose engagement data to the public. I randomly picked 5 malware families from each abused web app, resulting in 33 assets for my case study. For GitHub, WAE malware use benign and malicious repos, so I randomly picked 5 malware families for each case.

Table 2.6: WAE Engagement Information.

Family	Identity	Asset	FCD	LCD	Lan/Geo	Engagement		
Pastebin (Asset: Post)								
Neshta	vinmarcio	g3w5Zkzi	2022-01-01	2022-01-01	Alabanian	AllAssets: 3 AllViews: 6,142	Types: TXT(3) Visits: 6,099	AllFCD: 2022-01-01
Bymeria	Huynhnh92	Y8VWhxtG	2021-07-22	2021-07-22	English	AllAssets: 5 AllViews: 426,979	Types: TXT(3) Types:AutoIT(2)	AllFCD: 2019-09-19 Visits: 108,629
Ursu	pastwahman	xkkQYFEa	2018-01-09	2018-01-09	English	AllAssets: 47 AllViews: 5,389,380	Types: TXT(47) Visits: 793,381	AllFCD: 2016-11-11
Wacatac	sdfgsdf	qiMfPtmr	2021-12-21	2021-12-21	Russian	Visits: 3866		
Zusy	anthrax_	QKAdpkY	2021-12-19	2021-12-19	English	Visits: 796		
Github (Asset: Repo)								
Aauto	XTLS	Xray-core ¹	2020-11-09	2022-05-25	English	README: 99 Contributors: 39	Star: 8,499 Repos: 17	Fork: 1,448 Info: 84%
Apost	clangremlini	ayeloader-dll-repo ¹	2020-04-16	2022-02-03	English	README: 24 Contributors: 2	Star: 4 Repos: 13	Fork: 3 Info: 81%
Ayajbrci	keflu8	OpenFRE ¹	2022-01-16	2022-03-28	Canada	README: 13 Star: 3	Followers: 12 Repos: 50	Fork: 2 Info: 90.6%
Msilzilla	fireworm71	veriumMiner ¹	2014-04-23	2022-01-25	English	README: 274 Info: 71.8%	Star: 59 Followers: 8	Fork: 38 Repos: 4
Onlinegames	blaumaus	le_chiffre ¹	2020-11-14	2022-04-03	Ukraine	README: 45 Info: 87.5%	Star: 178 Followers: 35	Fork: 27 Repos: 32
Redcap	maks88192939	Programm	2020-09-28	2020-09-28	English	README: 0 Info: 71.8%	Followers: 49 Repos: 32	
Lazy	0x00BAD	st_mod	2020-05-25	2022-01-28	English	README: 0 Info: 78%	Fork: 2 Repos: 4	
Wacatac	aaleaf	Lite	2020-04-10	2022-11-16	English	README: 0 Info: 78%	Fork: 0 Repos: 9	
Tasker	max444432	RMS2	2021-09-09	2022-07-14	English	README: 1 Info: 71.8%	Fork: 0 Repos: 12	
Growtopic	aphony12	imcool	2021-06-27	2021-09-20	English	README: 2 Contributors: 1	Repos: 1 Info: 71.8%	
	deviluker	jszpy	2021-06-27	2021-06-27	English	README: 2 Contributors: 1	Repos: 1 Info: 71.8%	
	giganiga	nothing	2021-06-27	2021-09-20	English	README: 2 Contributors: 1	Repos: 2 Info: 71.8%	
	rosen6	rosesareroses	2021-06-27	2021-06-27	English	README: 2 Contributors: 1	Repos: 1 Info: 71.8%	
Telegram (Asset: Channel)								
Razy	h_money_1	h+money+1	2020-09-29	2020-09-29	English	Subscribers: 3	Views: 12	
Zenpak	johnyes13	johnyes1121	2020-12-20	2020-12-20	English	Subscribers: 3	Views: 6	
GDrive (Asset: File)								
Onlinegames	sahnoila24	hwid	2021-10-17	2021-10-17	English	Text: 76 Bytes		
Avemaria	reubzjohn	Egcs	2020-08-26	2020-08-26	Unknown	Binary: 574KB		
Netwiredrc	moham ²	Aqzm	2020-04-03	2020-04-03	Unknown	Binary: 815KB		
Remcos	Lorenz Kraft Formal D.O.O.	Bftv Ortl	2020-04-23 2020-07-13	2020-04-23 2020-07-13	Unknown	Binary: 378KB Binary: 1MB		
Sabsik	yuser20	XeroTool.a3x	2021-10-24	2021-10-24	Unknown	Binary: 85KB		
Blogspot (Asset: Web Page)								
Sisron	Nemorent	Blog Post	2012-02-05	2014-01-12	Egypt	Blogs: 230	Profile Views: 535	
Vobfus	nmournt	Blog Post	2012-02-05	2014-01-01	Egypt	Blogs: 116	Profile Views: 535	
Zpevdo	freechecking911	Blog Post	2018-07-29	2018-11-08	Thai	Blogs: 118	Profile Views: 14	
Virut	togpage	Blog Post	2012-12-09	2014-03-11	Egypt	Blogs: 2,380	Profile Views: 667	
Wordpress (Asset: Web Page)								
Symmi	turrona5	Blog Post	2011-05-14	2011-05-14	English	Blogs: 1		
Sysn	negragarchada	Blog Post	2011-07-01	2011-07-01	English	Blogs: 1		
Ayajbrci	tenah ³	Blog Post	2017-07-26	2017-07-26	Turkish	Blogs: 1		

1: Benign repositories hosting general-purpose programs that are fetched by WAE malware. See §2.5.4 for details. All other assets were manually verified as hosting malicious content. **2:** Short for mohamedshokry19792016. **3:** Short for tenahukame6823496829.

Surprisingly, the engagement data of malicious assets indicate that WAE malware could have infected up to 909,788 victims. Interestingly, most engagement data is from the view numbers exposed by Pastebin. Notably, this number is likely inflated by multiple accesses from the same sample and views from other sources like crawlers.

The engagement data results are presented in Table 2.6. Columns 4-5 show abused assets' First and Last Creation Date (*FCD* and *LCD*). Column 6 shows the geo-location (*Geo*) or language (*Lan*) of the abused assets derived using polygot [47]. Column 7 presents the engagement data supported by each web app. Rows 1-5 of Table 2.6 present the 5 Pastebin users and posts used by WAE malware, from which Marsea harvested the view number of the posts (*Visits*). Surprisingly, Marsea uncovered 908,109 views from 5 posts, averaging 181,621 views per post, indicating a significant extent of infection.

Benign & Malicious Assets. WAE malware also abuse benign web app assets (e.g., Xray-core Github repo used by Aauto, Verium cryptocurrency miner used by Msilzilla). To distinguish, given the extracted identity and assets, Marsea uses GitHub APIs to query the identity information and calculate the information completeness (*Info* in Column 7). Specifically, given all fields defined by the web app in the user class, U , and the fields filled by the identity, Id , Marsea calculates the information completeness using $Info = |Id|/|U|$. Similarly, by visiting the abused repositories, Marsea counts the lines of code (LoC) in the README .md file.

From Rows 6-18 of Table 2.6, I made several observations: (1) Column 7 shows that the average LoC of the README of benign repos and malicious repos are 91 and 1, respectively. Unlike benign projects that use README files, malicious repos do not require such comprehensive information. (2) Rows 6-16 show that the average information completeness (*Info*) of identities that create benign repositories and malicious repositories are 82.88% and 73.3%, respectively. Understandably, attackers prefer to leave this information blank during registration. Interestingly, Rows 13-16 of Table 2.6 show that

Table 2.7: Reporting To Web App Providers.

Collab Type	Web App Providers	Avg. Resp Time (days)	Reported	Taken Down
Bilateral	Discord	2	21	17
	GitbHub	103	27	1
	DuckDNS	2	1	1
	Afraid	2	1	1
	MediaFire	1	1	1
	Twitter	3	1	0
	Facebook	12	1	0
Unilateral	Pastebin	1	13	13
	Google	138	42	11
	WordPress	7	3	3
	Webs	3	1	1
	Tripod	4	1	1
	Blogspot	5	4	4
	Telegram	14	2	2
	Dropbox	11	10	10
Summary		24	129	65

attackers created 4 different GitbHub repos on the same day, which the same malware family abused. I suspect that weak registration verification encourages attackers to create multiple assets using different identities.

Unfortunately, my study found several web apps that do not collect engagement data or expose it to the public. Given the significant extent of WAE malware infection revealed by Marsea, I suggest the web app providers collect and provide engagement.

RQ4: By harvesting engagement data, Marsea revealed that WAE malware could have infected up to 909,788 victims from 33 abused assets. Marsea allows IRs to separate malicious and benign assets via identity completeness (82.88% complete for benign versus 73.3% for malicious identities) and asset comprehensiveness (91 average LoC for benign versus only 1 for malicious assets).

2.5.5 Novel Collaboration

The end goal of IR and web app collaboration is the remediation of web app abuse. During my study, Marsea identified 129 active malicious assets directly abused by 439 WAE

malware. Using the proof-of-abuse generated by Marsea, I collaborated with web app providers who have taken down 65 malicious assets (50%) to date. The detailed results of these collaborations are shown in Table 2.7. Unfortunately, as shown in Rows 2 and 9, GitHub and Google took 103 and 138 days to take action. Consequently, only 12 of the 69 live assets reported to GitHub and Google have been remediated. Of the remaining 60 reported assets on other web apps, 53 (83%) were taken down within 2 weeks. I continue to pursue the remediation of the remainder.

I categorize the collaboration into *bilateral* and *unilateral* in Column 1 of Table 2.7. Bilateral means web app providers update IRs throughout the remediation process and provide IRs with a response. This allows IRs to track the remediation process and assist web app providers when necessary. Unilateral means that IRs can only report the asset/identity as abusive via a ticket system. Reporting tools are often as simple as a button on the web app (e.g., Google does not provide any way for IRs to provide proof-of-abuse). Consequently, IRs are excluded from the subsequent remediation process. The directly abused assets I reported and took down are shown in Columns 4-5. Interested readers are directed to §A.2 for responses I received for bilateral collaboration. From Table 2.7, I observe the lengthy response period by more popular and reputable web app providers. For example, Google took 138 days to react. Even though unilateral collaboration can remediate the abuse relatively quickly, it can easily create hurdles for follow-up actions initialized by IRs if the remediation is slow. Therefore, rapid bilateral collaboration should be encouraged.

Lateral Remediation. As shown in Rows 1-3 of Table 2.6, besides the 3 posts directly abused by WAE malware, Marsea found 52 more posts from these identities. These additional assets give a unique opportunity to perform lateral remediation during my novel collaboration. As shown in Row 3 of Table 2.6, Marsea found the Ursu malware using the post (with paste key `xkkQYFEa`) by `pastwahman` on Pastebin. Enabled by the Pastebin API, Marsea can locate another 46 posts from `pastwahman`. The earliest creation date

spans 424 days before the *FCD* of the directly abused post. Marsea found that these potentially malicious posts have 5,822,501 views, 4,909,908 more than the directly abused posts. I collaborated with Pastebin to remove these assets from the same identity. Surprisingly, during my investigation of all posts from the `pastwahman` identity, I found the `DARKOM` (paste key `kE8tbd6t`) post with a VB script. The script queried a Pastebin post named `allinone` (paste key `q6dVaKSf`) under a different identity, `Getsilent`. Through further collaboration with Pastebin to remediate `Getsilent`, I took down 30 additional posts. Overall, starting from 1 post, lateral collaboration enabled me to take down an additional 76 assets.

Early Stage Remediation. During my study, Marsea observed WAE malware fetch additional malware from web apps. I also used Marsea on these dropped malware to rebuild the abuse chain and explore the possibility of early-stage remediation. Table 2.8 presents 3 first-stage WAE malware families with multi-stage abuse chains. Columns 1-2 of Table 2.8 present the malware families and the number of malware observed (Mal). Column 3 presents the abused web apps, and Column 4 shows the malicious vectors. Column 5 presents the attacker-controlled C&C server communication targets in each stage. Columns 6-8 present the number of web app sessions reconstructed (W), the number of malicious endpoints (M), and the percentage of malicious sessions (%M), respectively. I use $\hookrightarrow$ to present the dropper chain.

From Table 2.8, I observed that from 14 first-stage WAE malware (from 3 families), Marsea revealed 15 additional dropped malware from 6 families. More importantly, as shown in Column 8 of Table 2.8, the initial infections only abuse web apps (0% in the %M column). Later-stage malware increasingly rely on attacker-controlled C&C servers as the abuse chain gets longer (ending at 100% in the %M column). Ensuring the initial stage remains stealthy and persistent allows the attackers to update the assets on the web apps to drop new payloads once the later-stage malware are flagged as malicious. It is

Table 2.8: Multi-Stage Abuse Chain.

Family	Mal	Web Apps	Vec	Malicious C&C	W	M	%M
Dkomet ²	11	GDriveEXE Afraid	JS	- -	3	0	0%
↳ ⁴ Razy	2	GDriveEXE Ipify Discord Afraid	MF MP JS	mo00.com itvhacker.gq ¹	4	2	33%
↳ Zapchast	1	-	-	212.193.30.45 212.193.30.21	0	2	100%
↳ Disco	3	Discord Ipify	MP MF	itvhacker.gq ¹	2	1	33%
↳ Hosts	1	-	-	61.160.247.187	0	1	100%
↳ Dkomet ²	1	Discord Ipify Afraid	MP MF JS	mo00.com	3	1	25%
↳ Bdbd ³	1	Discord Ipify	MP MF	itvhacker.gq ¹	2	1	33%
Binder	2	GDriveEXE Ipify Discord Afraid	MF MP JS	-	6	0	0%
↳ Disco	2	Discord Ipify	MP MF	itvehacker.gq ¹	4	1	20%
↳ Dkomet ²	2	GDriveEXE Ipify Discord Afraid	MF MP JS	mo00.com	4	1	20%
↳ Disco	1	Discord Ipify	MP MF	itvhacker.gq ¹	2	1	33%
Wapomi	1	GDriveEXE Afraid	JS	-	3	0	0%
↳ Wapomi	1	-	-	ddos.dnsb8.net	0	3	100%

1,2,3: Short for itroblvehacker.gq, Darkkomet, Bladabindi.

4: Presents the payload dropping.

also interesting to note that 38% of malware dropped by GDrive connects back to GDrive to download different next-stage payloads, seeking persistent infection. This suggests a malware development toolkit abusing GDrive and Discord.

My findings show that IRs have a key opportunity: Early-stage remediation cuts this abuse chain, preventing WAE malware from dropping additional malware.

Migration Remediation. Marsea observed 4 cases of WAE malware migrating across different web apps or different identities under the same web app. I hand-verified the content

Table 2.9: WAE Malware Migration.

Family	Mal	<i>FSD</i>	Web Apps	Vector	Iden	Assets
Sohanad	2	2013-06	Webs	FD	se***3	settigs.xls
	1	2013-08	Webs	FD	ad***9	settings.doc
Sabsik	1	2021-06	GitHub	MF	cu***u	uattached
	8	2022-10	GitHub	MF	Ar***k	auth
Hynamer	2	2021-05	Github	EXE	sw***a	Nsудо.exe
	1	2021-09	Discord	EXE	88***2	Nsудо.exe
Msil	1	2022-01	Dropbox	MF	4g*****a	hwid.txt
	1	2022-01	GDrive	MF	sa*****a	hwid.txt

of the abused assets, and if the abused assets share the same content or format, I use it as proof of the migration. The results are shown in Table 2.9.

From Table 2.9, I made the following observations: (1) WAE migration can be divided into intra- and inter-web apps, as shown in Rows 1-4 and 5-8, respectively. (2) 100% of WAE malware in this case study changed identity names during migration. (3) 2 malware families (shown in Row 5-8), which account for 29.4% of the malware samples in Table 2.9, keep the same asset name during the migration. The attackers only update the assets name during intra-web app migration. Therefore, an information-sharing process among web app providers is needed to enable content-based asset monitoring.

RQ5: I reported 129 abused assets to web app providers, who took down 65 (50%) of them. I presented 3 novel collaboration opportunities. Marsea enabled me to identify web app abuse migration. From Lateral Remediation, Marsea found an additional 76 assets that I successfully remediated.

2.6 Related Work

Empirical Study of WAE Malware. Past research studied WAE as seen from the botnet [19], [21], [22], [48], [49] by detailing one or several specific attacks. Barroso et al. [50] presented a survey of cybercriminals exploiting public web servers to carry out different attacks. Zhang et al. [51] proposed an approach to detect compromised websites for searching poisoning attacks. Wang et al. [20] performed a comprehensive analysis of

Economic Denial of Sustainability (EDoS) attacks conducted through free public third-party services. Liao et al. [23] conducted a study on the abuse of cloud repositories as a malicious service by analyzing redirection chains. Li et al. [52] studied abusive hosts on Internet Service Providers (ISP). Alrwais et al. [53] studied the malicious activities which were hosted on the sub-allocations of the legitimate service provider. Botacin et al. [30] performed a longitudinal analysis of Brazilian Financial Malware, revealing the abuse of web apps. Han et al. [49] performed a large-scale analysis of malware interacting with Amazon EC2 [54]. Lever et al. [55] used the malware execution traces and the DNS queries to study the efficacy of network detection. While these studies focused on different attacks and analyzed cloud services, Marsea analyzes WAE malware to support live actions against malware. Our approach goes beyond investigating web apps to reveal unexpected abuse of non-standard web apps.

Malware Behavior Analysis. There are works using taint analysis [56], [57], API trace analysis [58], [59], [60], [61], [62], [63], and network traffic analysis [64], [65], [66] to reveal malware behaviors. However, Marsea uses symbolic data propagation to track the information flow and to perform multi-path exploration. Marsea bridges the gap between analyzed WAE malware and the abused web apps by attributing malicious vectors to the abused identities and assets. Further, Marsea harvests the engagement information of abused identities and assets to evaluate the extent of the infection.

Concolic Analysis. Several tools perform concolic analysis on a source code level [67], [68], [69] and a binary level [38], [70], [71], [72], [73]. Since these tools suffer from path explosion, the research community has several strategies to mitigate this problem [74], [75]. However, these tools are largely geared toward vulnerability analysis. There are also existing works built on top of S2E [40], [41], [76], [77]. Unfortunately, these works serve different goals from WAE malware analysis. Instead, Marsea uses a new code-orientated exploration strategy and introduces symbolic data through peripheral inputs and evasive functions. This

enables Marsea to categorize WAE through data flow analysis and will not face the challenge introduced by context-sensitive malware or logic bombs.

2.7 Discussion and Limitations

Ethical Discussion. I remediated WAE malware by collaborating with the web app providers. During the remediation process, I strictly followed the guidelines and terms of service provided by web app providers (e.g. [78], [79], [80], [81], [82], [83]) I *do not* store or use private credential information extracted from the malware to access any remote endpoints. I *do not* attempt to remediate the identities and assets WAE malware used without collaboration from web app providers.

Challenges of Concolic Analysis on Malware. Marsea is built upon concolic analysis. Like other concolic and symbolic analysis techniques [71], [84], [85], [86], Marsea can suffer from the path explosion motivating my design in §2.3.1. My validation in §2.4 empirically shows that Marsea avoids path explosion in WAE malware. Furthermore, malware often employ packing or obfuscation techniques. Since Marsea uses concolic analysis, the malware can unpack itself at runtime [32]. Prior works [63], [73], [87], [88] explored handling sophisticated malware deploying code obfuscation specifically aiming to thwart symbolic analysis. I followed their best practices while designing Marsea.

2.8 Conclusion

Marsea revealed a 226% increase in WAE-only malware since 2020, showing malware’s growing adoption of web app abuse. Marsea identified 893 WAE malware (97 families) abusing 29 web apps. Worse still, Marsea also found 430 WAE malware (48% of the 893 total) remained active with available web app assets. Assets for these WAE malware remained available for an average of 253 days. Enabled by Marsea, I reported 129 cases of abuse to web app providers, who took down 65 (50%) of them, and I continue to pursue the remediation of the remainder.

CHAPTER 3

VADER: PROACTIVE DETECTION AND REMEDIATION OF DEAD DROP RESOLVER MALWARE

Abstract

Dead Drop Resolver (DDR) malware evades traditional Command and Control (C&C) server takedowns by dynamically resolving C&C addresses hosted on popular web applications, such as Dropbox and Pastebin. These addresses are often manipulated (i.e., encoded or encrypted), rendering existing detection techniques largely ineffective. To tackle this challenge, I introduce VADER, a malware forensics system specifically designed for the proactive detection of dead drops. Analyzing a dataset of 100k malware samples collected in the wild, VADER identified 8,906 DDR malware samples from 110 families that leverage 273 dead drops across seven web applications. Additionally, it proactively uncovered 57.1% more dead drops spanning 11 web applications. Case studies revealed that over 40% of DDR malware samples employ sophisticated, layered de-manipulation algorithms, highlighting the prevalence and complexity of this evasion technique. Beyond detection, VADER enabled proactive remediation by discovering 13 previously unknown dead drops from a single DDR malware sample. This approach empowers web application providers to systematically scan their platforms, enabling the early detection and mitigation of dead drops.

3.1 Introduction

Dead Drop Resolver (DDR) is a technique developed by malware authors to evade traditional takedown methods for Command and Control (C&C) servers [89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99]. For example, Figure 3.1 shows the workflow of the *razy* [100] malware. *Razy manipulates* (i.e., encodes or encrypts) its C&C server address and hosts it

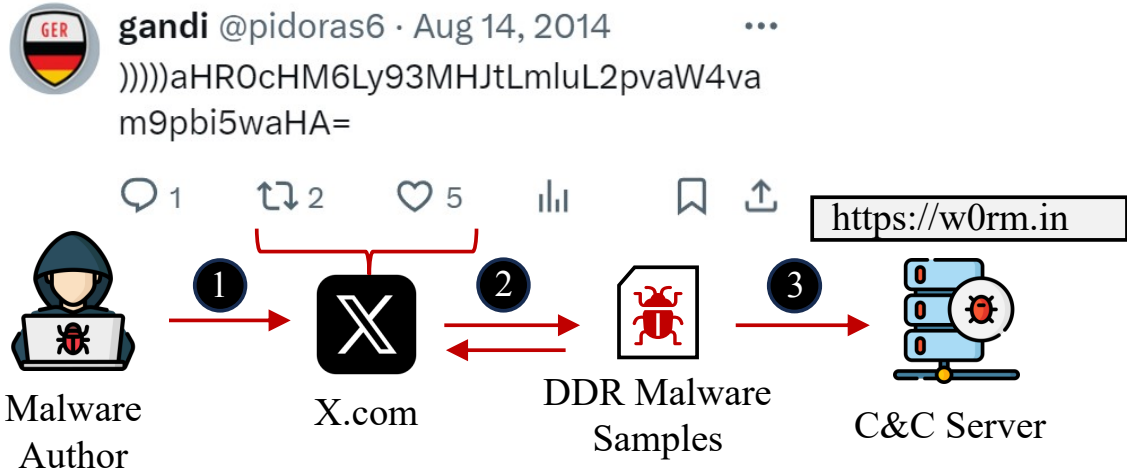


Figure 3.1: Razy’s [100] DDR workflow. ❶: The malware author *manipulates* the C&C server address and *posts* it on X.com as a dead drop. ❷: Razy executes on the victim systems and *fetches* the dead drop. ❸: Razy *resolves* the dead drop content to connect to the C&C server.

on X.com, serving as “dead drop” (❶). Once executed on the victim machine, *razy* fetches (❷) and decodes (❸) the dead drop content to resolve its C&C server address, confirming it as a DDR malware sample.

Current remediation strategies fail for DDR malware samples because web app providers respond to individual dead drops that a DDR malware sample is actively using (❸). However, malware authors can reuse manipulation techniques to encode new C&C addresses, creating polymorphic dead drops. As a result, the reactive approach misses opportunities to mitigate dead drops early in the DDR lifecycle (❶). This research proposes a proactive approach, where web app providers can scan their platforms to identify and neutralize dead drops.

Malware authors are increasingly exploiting web apps to host dead drops [1], yet state-of-the-art techniques lack the scalability needed for proactive remediation by web app providers. The current methods primarily rely on manual inspection to understand how malware authors encode C&C addresses hosted on web apps [101], [102], [103] or stored as blockchain transactions [104], [105]. Proposals such as submitting new transactions to update the most recent blockchain data to sinkhole C&C addresses [106] demand extensive manual effort and remain ad-hoc. Similarly, web app providers can try to detect dead drops

by profiling incoming connections from DDR malware samples to their platforms. However, existing malicious connection detection techniques [55], [89], [96], [107], [108], [109], [110] are ineffective in the DDR context. This is because web apps enforce the use of specific protocols for all connections, making it impossible to distinguish between legitimate and malicious connections.

Adding to these difficulties, web app providers face technical challenges in combating DDR malware. One key issue is the wide variety of manipulation techniques (e.g., BASE64), making it impractical to decode all content using every method. Even worse, C&C addresses are often encoded using multiple layers of techniques, some requiring specific parameters (e.g., XOR). In fact, single dead drops can employ three or more manipulation layers (§3.5.4), rendering brute forcing infeasible. However, my analysis of DDR incidents shows DDR malware samples must be equipped with the precise combination and configuration of de-manipulation techniques. This led to my first key insight: Understanding how DDR malware samples decode dead drop content allows web app providers to derive reusable “recipes” to proactively identify similarly encoded C&C addresses.

Before extracting recipes, the immediate challenge web app providers face is confirming a malware sample’s DDR capability. This is further complicated when a DDR malware sample connects to multiple web apps and C&C servers, creating intertwined and ambiguous connections. Rather than focusing solely on these connections alone, my detailed analysis of DDR malware samples reveals distinct operations for fetching, de-manipulating, and establishing new C&C connections. This led to a second key insight: Analyzing the information flow, starting from the fetched content, effectively isolates and confirms DDR logic.

Even with the DDR capability confirmed, identifying the exact de-manipulation techniques and their order requires manual analysis [104], [106], [111]. Nevertheless, my research discovered a third key insight: De-manipulation routines applied to fetched content can reduce to mathematical operations. By abstracting these formulas and comparing them

to known patterns, web app providers can confirm the de-manipulation techniques used and generate effective recipes for proactive remediation.

To explore my key insights, I studied DDR malware samples with support of Netskope, the leading Secure Access Service Edge (SASE) provider, which delivers cloud security and networking to more than 30% of Fortune 100. I developed VADER for proactive dead drop resolution and remediation. VADER is a web app and malware analysis framework to automatically detect and proactively uncover dead drops. VADER systematically explores a DDR malware sample to *localize the DDR logic* (§3.3.1). It then identifies de-manipulation algorithms by analyzing the symbolic expressions generated during its execution. However, a significant challenge lies in accurately isolating the relevant segments of these expressions for precise matching. To address this, VADER employs an innovative *Input/Output (IO) boundary isolation* technique (§3.3.2), that delineates the boundaries of routines within the symbolic expressions. This approach enables VADER to effectively isolate and match expressions, deriving de-manipulation recipes (§3.3.3). Through this recipe-based pattern-discovery approach, VADER can efficiently uncover C&C addresses concealed within dead drops.

I deployed VADER on 100k malware samples from 2012 to 2022, identifying 8,906 DDR malware samples and 273 dead drops. Pastebin was the most prevalent platform, accounting for 68% of cases. This underscores the importance of VADER-generated recipes for proactive dead drop discovery. As of this submission, I have removed 94.4% of identified dead drops through collaboration with web app providers, disrupting at least 6,674 DDR malware samples by eliminating the dead drops they relied on. Applying these recipes to network traffic datasets from open-source projects [112], I uncovered four additional abused web apps and 72 previously undisclosed dead drops. I also presented two case studies on de-manipulation recipe complexity and a real-world forensic analysis of a prominent remote access trojan. The VADER framework can be found at: <https://github.com/CyFI-Lab-Public/VADER>.

3.2 Overview

3.2.1 Running Example - The Mudrop Malware

In VADER's practical demonstration to combat DDR malware and uncover more dead drops, I use *mudrop* as the running example. Imagine this forensic scenario: the WordPress [113] security team receives an alert from FIRST [114], a global cybersecurity organization, about a new malware sample, *mudrop*. Flagged as a potential threat to WordPress, *mudrop* is shared by FIRST as part of its collaborative security effort.

Upon analyzing the sample, WordPress identifies that *mudrop* connects to WordPress at `blizzartone.wordpress.com` and `http://188.190.98.163/configs`, a potential C&C server.

At this point, WordPress must answer several forensic questions:

Q1: Is *mudrop* a DDR malware sample?

Q2: Does *mudrop* use WordPress as a dead drop?

Q3: How does *mudrop* decode its dead drop?

Q4: Are there other dead drops hosted on WordPress?

Unfortunately, answering these questions poses significant research challenges. Firstly, there is no definitive proof that *mudrop* relies on WordPress to resolve its C&C server address. *Mudrop* can connect to WordPress to test internet connectivity before executing its malicious payload, as many malware samples do. Second, if it is a DDR malware sample, and the content is retrieved from `blizzartone.wordpress.com`, it will be encoded, as shown below:

```
f237769666e6f636f2336313e28393e2039313e2838313f2f2a307474786
```

Clearly, this encoded snippet does not match the C&C address, complicating efforts to confirm DDR behavior.

Table 3.1: VADER’s Mudrop Proactive Discovery Results.

VADER’s <i>Mudrop</i> Analysis Output	
Dead Drop	http://blizzartone.wordpress.com
↳Dead Drop Content	f237769666e6f636f2336313e28393e2039313e2838313f2f2a307474786
↳Recipe	Base16 Decoding + Char Rotation x4
↳C&C Address	http://188.190.98.163/configs
Proactive Discovery 1	
Dead Drop	http://selfcut.wordpress.com
↳Dead Drop Content	336313e28393e2039313e2838313f2f2a307474786
↳C&C Address	http://188.190.98.163
Proactive Discovery 2	
Dead Drop	https://brainbot02.wordpress.com
↳Dead Drop Content	F247F626F2137313E2836313E2134323E22393F2F2A307474786
↳C&C Address	http://92.241.168.171/bot
Proactive Discovery 3	
Dead Drop	https://suck4.wordpress.com
↳Dead Drop Content	131323e2538313e2037313e2733313f2f2a307474786
↳C&C Address	http://137.170.185.211

To address these research challenges and answer the aforementioned questions, VADER employs novel techniques to: (Q1 & Q2) *localize* DDR logic within the malware (§3.3.1); (Q3) apply de-manipulation to dead drop content (§3.3.2); and formulate a de-manipulation *recipe* (§3.3.3). (Q4) Then, the recipe can be applied to web app content or network traffic, proactively identifying *previously undisclosed* dead drops. This multiplies the effectiveness of DDR remediation—a contribution not yet explored in prior research. Moreover, WordPress can share *mudrop*’s recipe with FIRST and web app providers to extend remediation.

Once the *mudrop* sample is provided, VADER identifies its DDR logic, notifying the WordPress provider that the *mudrop* sample retrieves dead drop content from a WordPress post and decodes it to obtain the C&C address, as shown in Table 3.1. The provider can then remove the dead drop content. However, providers still lack insight into how *mudrop* decodes the content, which is key to proactively finding hidden dead drops. By analyzing the *mudrop*’s DDR logic, VADER tracks the decoding process, revealing layered Base16 encoding and character rotation algorithms (i.e., the recipe) to extract the dead drop content (see Table 3.1, *VADER’s Mudrop Analysis Output*).

3.2.2 Proactive Dead Drop Discovery

VADER’s recipe goes beyond simply validating a malware’s dead drop. It enables content-based scanning (e.g., web apps, network traffic) to proactively uncover polymorphic dead drops. For example, WordPress could scan its platform for identical content and remove known dead drops. However, as C&C servers rapidly evolve, attackers can abuse WordPress to host varied dead drops that resolve to different C&C addresses. This makes it nearly impossible for WordPress to locate these dead drops by matching identical content.

To overcome this challenge, VADER scans accessible WordPress posts or messages, decodes them using the de-manipulation recipe from *mudrop* (Table 3.1, Row 4), and extracts IPs/URLs via a regular expression. It then checks these against blocklists (e.g., VirusTotal, URLHaus) to classify web app accounts as dead drops for remediation. As Table 3.1 illustrates, VADER proactively discovered three previously unknown WordPress dead drops, *selfcut* (Proactive Discovery 1), *brainbot02* (Proactive Discovery 2), and *suck4* (Proactive Discovery 3), demonstrating its ability to detect varied content beyond exact matches.

3.2.3 VADER in Practice

Web App Providers + VADER. Section §3.2.2 shows how VADER’s recipes can enhance remediation by proactively uncovering dead drops. While my lab setting limits access to web app platforms, providers are better positioned to scan their own platforms at scale. Platforms like Discord already scan hosted files proactively [115], and Google uses its Content Safety API to detect child sexual abuse material at scale [116]. Additionally, Roblox, Discord, and Reddit also work with law enforcement to identify extremist users via advanced scanning [117]. I am therefore confident that these providers can use VADER to proactively detect dead drops.

Intrusion Detection System + VADER. Intrusion Detection Systems (IDS) can integrate

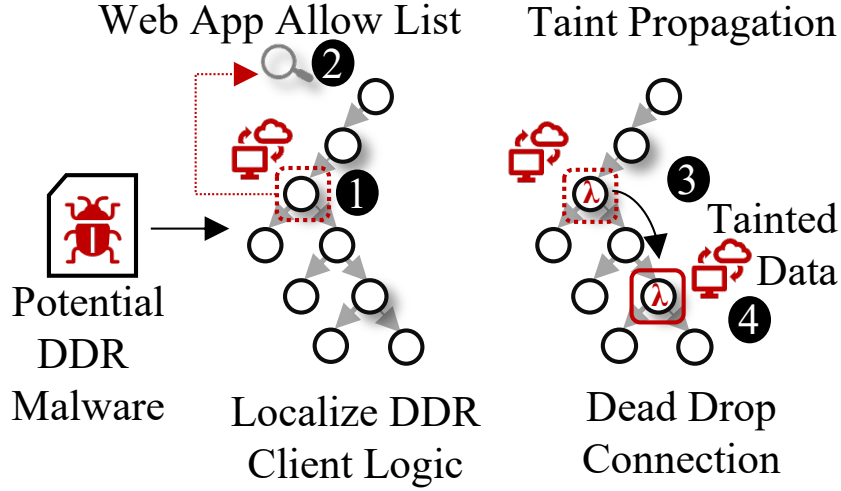


Figure 3.2: DDR Logic Localization.

VADER. When malware is detected, VADER verifies its DDR logic and extracts recipes, which IDSs can use to analyze network traffic and de-manipulate embedded payloads. If any IPs or URLs are found, analysts can use internal tools to confirm maliciousness [118], enabling proactive dead drop discovery and strengthening network protection. This is shown in §3.4.2, where VADER’s recipes were applied to network traffic. By using these recipes to de-manipulate payloads, I proactively uncovered 72 dead drops across 11 web apps, hiding 67 C&C addresses. My collaboration with web app providers led to the removal of 94.4% of these proactively identified dead drops.

3.3 Methodology

VADER uses a three-phase approach: *DDR logic localization* (§3.3.1) takes a potential DDR malware sample as input to confirm DDR capabilities. The output is a symbolic expression generated during DDR malware sample exploration. VADER then *isolates de-manipulation boundaries* (§3.3.2) to segment the symbolic expression for comparison with a set of reference de-manipulation algorithms for *recipe* identification (§3.3.3). Finally, a *recipe-based pattern-discovery* solution demonstrates how the recipe can be used to proactively uncover previously undisclosed dead drops.

3.3.1 DDR Logic Localization

The goal of this phase is DDR capability identification via logic localization. To localize DDR logic, VADER uses concolic (concrete and symbolic) analysis to first identify the *DDR Client Connection* and then the *Dead Drop Connection*.

DDR Client Connection. During VADER’s analysis, VADER intercepts the DDR malware sample’s invocation of network APIs used for web app connection (Figure 3.2, ①). Web app connection is determined by comparing the connection target (e.g., *mudrop* uses `wordpress.com`) against a predefined list of allowed web app candidates (Figure 3.2, ②). This list is based on Tranco [45]¹, allowing me to identify only benign websites that support DDR². After the web app connection is established, the DDR malware sample retrieves the encoded content from the dead drop (e.g., `blizzartone` for *mudrop*).

Dead Drop Connection. To confirm the dead drop connection, VADER injects symbolic data into the memory location allocated by the malware to store the fetched content (e.g., *mudrop* uses `lpBuffer` of `WinHttpRequestData`) and tags it (Figure 3.2, ③). VADER uses concolic taint propagation to track the tag. However, de-manipulation algorithms generally contain loops that result in multiple states for exploration, thereby increasing computational complexity when using concolic execution. De-manipulating the encoded content requires that each character be transformed into cleartext. Thus, the content length (e.g., `lpdwNumberOfBytesRead`) determines how many iterations of character transformation are necessary. To ensure tractable analyses amid multi-state exploration, VADER uses state prioritization. Specifically, VADER prioritizes unexplored code and selects states corresponding to new code regions. For example, if VADER explores a de-manipulation loop, it will generate two states:

(1) $i < \text{sizeof}(\text{lpdwNumberOfBytesRead})$

¹Available at <https://tranco-list.eu/list/7X9VX>.

²I assume malicious connection targets if not found on Tranco.

Table 3.2: Common Data Manipulation Algorithms.

Decryption & Decoding Algos References	
Exclusive OR (XOR)	[119], [120], [121], [122], [123]
AES	[124], [125]
DES	[126], [127]
Data Protection API	[127]
RC4	[119]
String to Int, Int to String	[128]
Character Rotation	[121], [122]
Character Subtraction	[122]
Base 16	[101], [122], [129]
Base 32	[130]
Base 64	[100], [102], [121], [122], [129]
Base 85	[131], [132]
String Parsing	[100], [101], [121], [128], [133]

(2) $i > \text{sizeof}(\text{lpdwNumberOfBytesRead})$

As execution continues in the first state, two additional states are generated:

(1₁) $i + 1 < \text{sizeof}(\text{lpdwNumberOfBytesRead})$

(1₂) $i > \text{sizeof}(\text{lpdwNumberOfBytesRead})$

Since state (1) and (1₂) explore the same code in two different states, state (1₂) is de-prioritized. This technique ensures that VADER can efficiently handle loops while ensuring that de-manipulation algorithms are fully explored as it prioritizes state (1) until complete.

During exploration, the tag emerges in a network connection API (e.g., *mudrop* uses `pswzServerName` for `WinHttpConnect`) Figure 3.2, ④. Importantly, what remains is a symbolic expression with all data computations performed on the manipulated content.

3.3.2 De-Manipulation IO Boundary Isolation

This phase aims to isolate de-manipulation algorithms in the symbolic expression using their input and output data to mark their boundary. De-manipulation includes decryption (e.g., AES) and decoding (e.g., Base16). My preliminary investigation suggests 14 de-manipulation (nine decoding, five decryption) algorithms common in malware (Table 3.2).

Reference Algorithm Implementations. I leveraged open-source repositories and libraries to find algorithm implementations, as detailed in Table 3.2. VADER supports extensions by adding more algorithm source code. These implementations are (1) directly integrated into VADER for input-output observation or (2) symbolically explored to generate expressions that represent their computational processes. VADER uses (1) to identify decoder input-output boundaries and (2) to validate de-manipulation algorithms against these boundaries.

Symbolic expressions, especially when subjected to malware-induced mathematical manipulations, lack clear observable boundaries, complicating the partitioning of multiple de-manipulation algorithms and potentially hindering dead drop discovery. To address this, VADER employs the symbolic expression from DDR logic localization (§3.3.1) and uses concrete decoder analysis (§35) and decryption source-to-sink mapping (§35) to isolate decoders and decryption processes.

Concrete Decoder Analysis

To isolate decoder boundaries, VADER uses Algorithm 2, which relies on concrete values computed during DDR malware sample execution (concrete values are based on constraints accumulated during execution) to find the input and output (IO) of the decoder. As VADER analyzes the DDR malware sample, Algorithm 2 evaluates memory accesses and concretizes its symbolic contents (Line 4 to Line 6). Next, it iterates through decoders from Table 3.2 using the concrete data as input (Line 7 to Line 8). Each output is stored in a container for algorithm-to-instruction mapping L for later referencing (Line 9). Thus far, the concretized data C (Line 6), the decoder D (Line 7), the decoded result d (Line 8), and L (Line 9) have been initialized. This process continues throughout VADER’s analysis. Concurrently, the algorithm iterates through all previously stored results in L (Line 10 to Line 13) to extract comparative data for boundary isolation. Specifically, if the algorithm finds a previous decoded value (e.g., the C&C address) that matches the current concrete value, then the

Algorithm 2: De-Manipulation Boundary Isolation

Input: *Malware*, *Decoders\Decryptors* = $\{D^1 \dots D^{16}\}$
Output: *DB* : Container to store De-Manipulation Boundaries

```
// Store Algorithm-to-Instruction mapping
1  $L \leftarrow \emptyset$ 
2 Function DeManipulationBoundary (Malware)
3   while Instruction  $\leftarrow$  ExploreMalware(Malware) do
4      $I \leftarrow$  Instruction
5     if  $M \leftarrow$  MemoryAccessed(I) then
6        $C \leftarrow$  ConcretizeMemoryContent(M)
7       // Iterate 9 Decoder Algorithms
8       foreach  $D \in$  Decoders do
9          $d \leftarrow$  DecodeConcreteData(D, C)
10        UpdateDecoderMapping(L, I, C, d)
11        // Iterate previously mapped results
12        foreach  $l \leftarrow L$  do
13           $I_p, C_p, d_p \leftarrow l$ 
14          // Previously decoded results equals current
15          // concrete value
16          if  $d_p \equiv C$  then
17            UpdateBoundary(DB, D, Ip, I)
18          end
19        end
20      end
21    else if  $C_i \leftarrow$  ImportedFuncCall(I) then
22      // Iterate 7 Decryption Algorithms
23      foreach  $D \in$  Decryptors do
24        // Locate Start and End boundaries
25        if  $S, I_s \leftarrow$  DecryptIO(D, Ci) then
26           $I\text{Boundary} \cup (S, I_s)$ 
27        else if  $E, I_e \leftarrow$  DecryptIO(D, Ci) then
28           $O\text{Boundary} \cup (E, I_e)$ 
29        end
30      end
31    end
32  end
33  // Create Decryption IO Pairs
34   $\text{DecryptIOPairs} \leftarrow$  MatchIO(IBoundary, OBoundary)
35  for  $Pr \in$  DecryptIOPairs do
36     $D, I_s, I_e \leftarrow Pr$ 
37    UpdateDeManBoundary(DB, D, Is, Ie)
38  end
39  if DB then
40    ClosestBoundary(DB)
41  end
42 end
```

decoder boundary begins at the instruction of the decoded value to the instruction of the current matching concrete value.

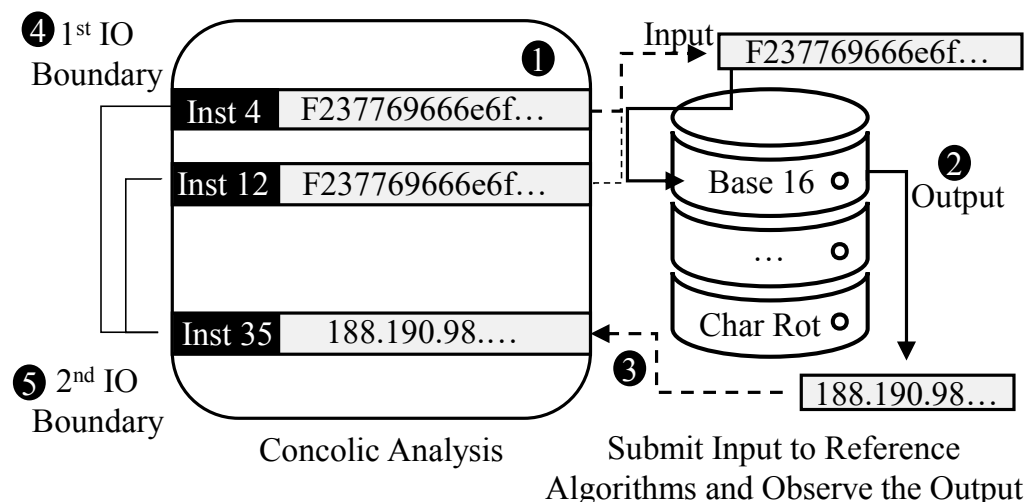


Figure 3.3: *Mudrop*'s Decoder IO Boundary Isolation.

I illustrate Algorithm 2 using *mudrop* in Figure 3.3. At instruction #4 (*Inst4*), *mudrop* accesses memory and VADER concretizes the memory to *mudrop*'s fetched content (Figure 3.3, ①). The input is submitted to reference decoder algorithms, and the output (Figure 3.3, ②) is stored for later comparison. As execution continues, VADER compares concretized results with previously decoded results to identify a match. If a match is identified (Figure 3.3, ③), the decoder boundary (e.g., Base64) is from *Inst4* to *Inst35* (Figure 3.3, ④). However, VADER may identify multiple boundaries for each decoder. Thus, the boundary for Base64 is also *Inst12* to *Inst35* (Figure 3.3, ⑤). This occurs if a wrapper function is used that does not modify input or output values. Before completing boundary analysis, VADER computes the shortest distance between boundary instructions for boundary isolation (Line 33 in Algorithm 2).

Using the boundary, VADER segments the symbolic expression from §3.3.1 for verification. An astute reader may consider symbolic expression matching redundant since VADER has already concretely delineated the DDR malware sample's decoders. This is necessary to discriminate two algorithms producing the same output, e.g., if *mudrop*'s encoded content

```
f237769666e6f636f2336313e28393e2039313e2838313f2f2a307474786
```

is decoded with Base16 and character rotation, the result is `http://188.190.98.163/configs` (see Table 3.1). If we use the same input and XOR with

```
f25f02e216dcd919c30b5b3dd3baa3cc3aab3dd3b503c919dcd612e20f
```

we arrive at identical outputs. If the DDR botnet orchestrators post additional encoded messages, relying on only the concretely derived decoders could lead authorities to apply the incorrect de-manipulation recipe. Thus, VADER compares each segmented boundary with its corresponding reference implementation symbolic expression for robust verification.

Decryption Source-To-Sink Mapping

VADER intercepts invoked API for DDR logic localization (§3.3.1). These interceptions also prove useful to link decryption APIs-in-sequence for analysis. Specifically, while VADER explores the DDR malware sample (Line 3 in Algorithm 2), it evaluates call instructions and selects those that are specific to imported functions (Line 17 in Algorithm 2). If the imported function is an API responsible for instantiating decryption (e.g., `CryptAcquireContext`), VADER updates the *IBoundary* set representing the boundary source (Line 19 to Line 20 in Algorithm 2). If the boundary sink is identified (e.g., `CryptReleaseContext`), VADER updates *OBoundary* (Line 21 to Line 22 in Algorithm 2). To pair sources and sinks, VADER uses *MatchIO()* (Line 27 in Algorithm 2). For example, during concolic exploration, if two states are explored, and the first state invokes a source and sink API, those APIs are paired since they were invoked and intercepted by VADER from the same state. This ensures that APIs invoked in different paths are not incorrectly paired. Finally, these pairs are used to update my container of all de-manipulation boundaries for analysis (Line 28 to Line 28 in Algorithm 2).

Algorithm 3: Symbolic Expression Matching

Input: DB : Container to store Decoder Boundaries**Output:** DR : Decoding Recipe

```
1  $M_\lambda \leftarrow$  Malware Symbolic Expression
2  $RI \leftarrow$  Symbolic Expressions for Reference Implementations
3 Function SymbolicExpressionMatching( $DB$ )
4   foreach  $b \in DB$  do
5     // Extract the decoder and start & end boundary addresses
6      $D, I_s, I_e \leftarrow b$ 
7      $s \leftarrow \text{SegmentExpression}(M_\lambda, I_s, I_e)$ 
8     // Assign decoder symbolic expression
9      $ri \leftarrow RI.index(D)$ 
10    if  $dm \leftarrow \text{CompareExpressions}(s, ri)$  then
11      |  $DR.add(dm)$ 
12    end
13  end
14 end
```

3.3.3 De-Manipulation Recipe Identification

This phase develops the de-manipulation recipe using symbolic expression matching. Encoded content may include extra string markers or inconsistently manipulated data. Without segmenting the symbolic expression before comparison with a reference implementation, false negatives can occur. Therefore, granular segmentation is essential for accurate verification of de-manipulation algorithms.

Segmenting Expressions. Algorithm 3 takes DB (decoding boundaries) from Algorithm 2 to segment the symbolic expression (lines 4-6). Each boundary b (lines 4-5) contains the decoder type D and the starting I_s and ending I_e boundary addresses used to segment the expression based on its attributes, i.e., addresses where the symbolic expressions grow. VADER parses only those parts of the expression within the boundary (Line 6 in Algorithm 3). Now, VADER compares the segmented expression with the symbolic reference implementation expression corresponding to D (Line 7 in Algorithm 3).

Compare Expressions. VADER injects symbolic data λ to localize the DDR logic, which generates a malware symbolic expression M_λ (Figure 3.4, ①). To generate the reference

implementation expressions, M_λ is used as input to decoding algorithms (Figure 3.4, ②), ensuring the resulting expression (e.g., $B16_\lambda$ in Figure 3.4, ③) assumes the constraints imposed during the malware execution. This is a prerequisite for symbolic expression comparison, which relies on symbolic solvers. When a decoding algorithm is symbolically explored, concrete values assumed during forking correspond to the previous constraints imposed by M_λ .

Now, having both symbolic expressions, VADER can compare them for equivalence (Figure 3.4, ④). For comparison (Line 8 in Algorithm 3), there are two conditional constructs: (1) check whether the overall expressions match; if not, (2) solve for and compare the concrete outputs. Toward clarity, I use a symbolic expression with starting λ values (`Read byte_1, v0_xor_0`). This expression is used as input to two versions of an XOR by `0x23` algorithm. As the expression is decoded, it grows with additional operations corresponding to algorithmic computations. For example, one byte of M_λ and XOR_λ is partially transformed into:

$$M_\lambda = (\text{Or } (\text{ZExt } (\text{Read byte_1, v0_xor_0})) \text{ 0x23})$$

$$XOR_\lambda = (\text{Xor 0x23 } (\text{ZExt } (\text{Read byte_1, v0_xor_0})))$$

Comparing M_λ and XOR_λ considers node placement, edges, and expression size. This static check is less computationally expensive. If it fails, VADER invokes the symbolic solver to compare the concrete outputs of both expressions. Concrete values are based on the constraints from M_λ that are imposed on XOR_λ during its execution. This ensures that expressions are evaluated based on the same constraints, resulting in the same concrete output if they are functionally identical. Since each comparison is per explored path, the results are a ratio of matches versus non-matches, i.e., when concretized outputs for each path are equal, it is a match; otherwise, it is a non-match. If the ratio is high enough (explained in §3.5), their equivalence is confirmed (Figure 3.4, ⑤).

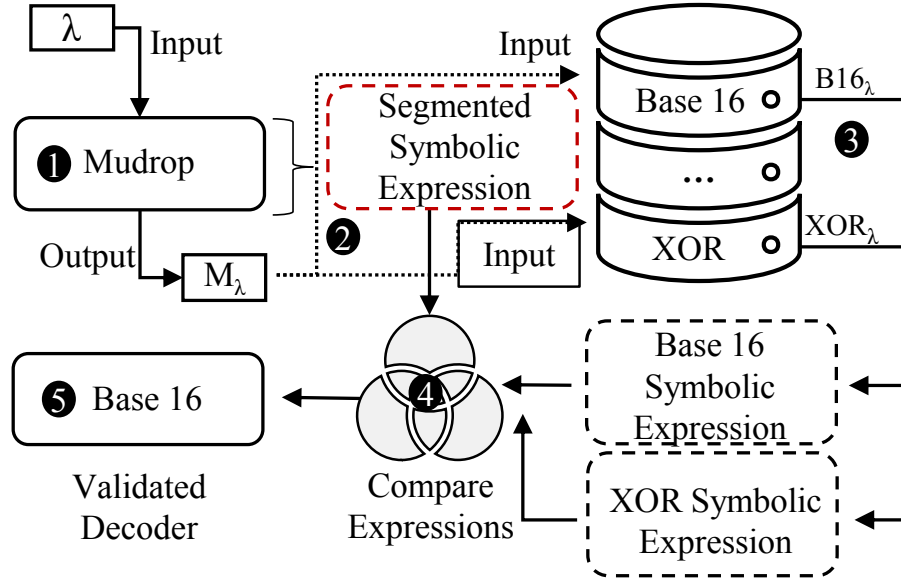


Figure 3.4: Symbolic Expression Matching For *Mudrop*.

Compare Decryption Expressions. At this stage, VADER verifies decryption algorithms by partitioning boundaries based on source and sink markers. These boundaries are set by analyzing the constraints on sinks (*OBoundary*) in relation to their sources (*IBoundary*). It is important to recall that VADER employs taint analysis during DDR logic localization (§3.3.1). Figure 3.5 illustrates segmenting symbolic expressions with isolated boundaries (note: No discovered DDR malware sample encrypts dead drop content; this example uses a non-DDR malware sample to demonstrate the approach). The dotted lines represent the matching of input [in] constraints from a parameter with the output [out] constraints of at least one API predecessor. When a decryption API, like `CryptReleaseContext`, is used, VADER conducts a backward analysis of constraints to trace the API sequence, confirming the decryption algorithm. This *decryption constraint-chaining* method links the constraints of recovered APIs to their preceding APIs.

In Figure 3.5, decryption starts after data retrieval from the web app (1), followed by cryptographic context initialization (2) and key derivation (3) before data decryption (`lpBuffer`, 4). VADER uses backward analysis to identify decryption constraints and the sequence of APIs in the malware’s symbolic expression, extracting only the relevant

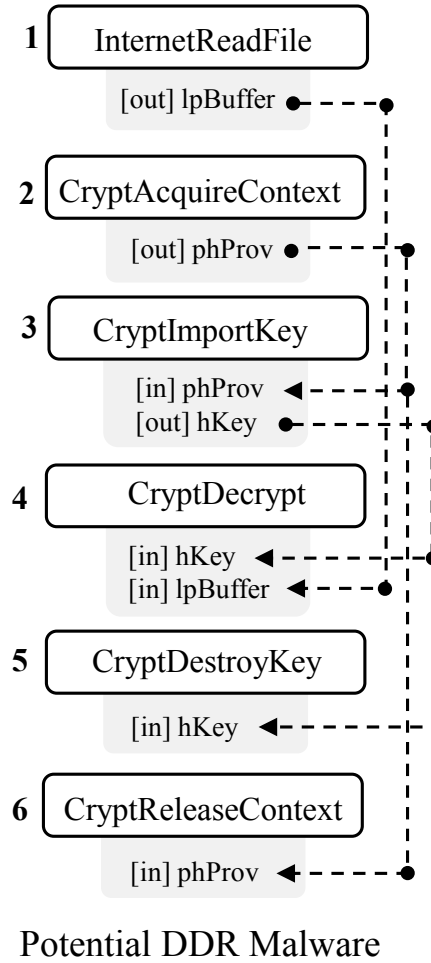
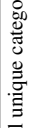


Figure 3.5: Constraint Chaining For Crypto Identification.

decryption portion.

To create reference symbolic expressions for comparison, VADER uses decryption models based on the API sequence from Table 3.2. By examining these implementations, VADER generates a symbolic expression and applies decryption constraint chaining to identify constraints related to the API sequence. The segmented expression is then compared with reference constraints using the *compareExpression* function (Algorithm 1, line 8). A match confirms that the decryption API computations align with the reference models. Upon matching, VADER produces a de-manipulation recipe detailing the decryption steps used by the DDR malware sample. This recipe can be applied to web app content to determine whether it represents a dead drop.

Table 3.3: Distribution Of DDR Malware Found In Our Study.

Web App	Domain	# M ¹ Dead Drop	Malware/Year 2012 - 2022	First Seen	Last Seen	# F ¹ Dead Drop	Backup Dead Drop	Total Recipes	Unique Recipes
PasteBin	pastebin.com	6,053		2014-02	2022-06	32	0	6,053	3
Blockchain	blockchain.info	776		2017-12	2019-11	59	111	776	1
	blockcypher.com	573		2017-12	2019-11	41	415	573	1
	bitaps.com	617		2017-11	2021-10	16	423	617	1
	coinmarketcap.com	45		2017-12	2019-11	10	45	45	1
	blockr.io	200		2017-12	2021-06	5	200	200	1
Blockchain Aggregate		2,200		2017-11	2021-06	81 ²	1,194	2,200	1
X	twitter.com	473		2017-10	2022-03	8	3	473	2
Google	docs.google.com	9		2018-02	2022-12	1	1	9	1
	googleusercontent.com	137		2017-06	2020-06	8	0	137	1
	drive.google.com	9		2018-02	2022-12	1	1	9	1
Dropbox	dropbox.com	11		2017-06	2020-06	2	0	11	1
GitHub	github.com	7		2017-11	2018-10	2	0	7	1
WordPress	wordpress.com	4		2012-09	2014-08	1	0	4	1
Total		8,906		2012-09	2022-12	110 ²	1,199	8,906	7 ²

1: M (Malware) and F (Families), 2: This is not the total column sum but the total unique categories.

3.4 Post-Deployment Findings

I deployed VADER on a 100k malware dataset built by randomly downloading Windows executables uploaded to VirusTotal between 2012 and 2022 [31]. Each malware sample triggered more than one anti-virus (AV) engine, based on dataset parameters from Zhu et al. [134], who studied VirusTotal labeling dynamics. Tranco [45] ensured the malware connected to at least one web app.

I present the distribution of DDR malware and its web apps in Table 3.3. For intrusion detection, I applied VADER’s methods to real-world network traffic from Netresec [112] (Table 3.4), a source cited in top-tier research [135], [136]. For HTTPS, I accessed the web app endpoint, decoded the response with the recipe, and extracted IP/URL addresses, which I scanned on VirusTotal [31] and URLhaus [137] to confirm maliciousness. Finally, I provide two case studies to demonstrate the prevalence and complexity of de-manipulation recipes and a practical approach to proactively discover dead drops.

3.4.1 DDR-Enabling Web Apps

Table 3.3 shows the distribution of DDR malware. Column 1 lists web apps, Column 2 their domains, and Column 3 the number of malware samples (#M) using each app. Column 4 shows unique dead drops, while Columns 5–7 provide a temporal snapshot over 11 years, using first- and last-seen dates from VirusTotal. Column 8 reports distinct malware families (#F), and Column 9 highlights backup dead drop domains, used if the primary fails. Columns 10–11 list de-manipulation recipe counts and unique recipes discovered.

From Table 3.3, Pastebin accounts for about 68% of discovered DDR malware (6,053). VADER found 44 unique dead drops (22% of all) across 32 malware families. The cases of Pastebin DDR malware that I identified span 2014-2022, with a major spike in 2021-2022, accounting for nearly 5k discoveries. Historically, Pastebin has hosted stolen content or malware [8]. This work is among the first to expose its pervasiveness as a platform for

hosting dead drops. VADER identified one recipe per malware and four unique recipes for Pastebin DDR malware (Columns 10–11). The low count of unique recipes relative to malware volume shows VADER is scalable for proactive detection.

Surprisingly, Blockchain web apps are the second most used for dead drops, spread across five domains. The most common, `blockchain.info`, hosts 776 malware samples retrieving Txn or wallet IDs and de-manipulating them to C&C addresses. VADER found 37 Blockchain dead drops (24 Txn IDs, 13 Wallet IDs) and 2,200 samples (17% of dead drops, 25% of malware). Column 9 shows 1,194 Blockchain samples with backups, e.g., `blockr.io` as primary and `blockchain.info` as backup. Both support querying Txn records using one recipe (Column 10). While Blockchain records are immutable, dead drops can still be flagged when Txn values match dead drop content.

VADER also found 11 X.com dead drops in 473 malware. I classified dead drop origins as (1) hard-coded, like traditional DDR, and (2) dynamically generated, similar to DGA malware. This emerged in the sample *fugrafa*, which uses X.com to host encoded C&C addresses. VADER found three backup X.com dead drops during DDR logic localization. Further analysis showed that the account name varies based on a function’s output, generating accounts like `np8j7ovqdl`, `q5euqysfu5`, and `qistp743li`. Despite this complexity, VADER, combined with DGA counteraction techniques, can effectively mitigate such threats [138], [139].

VADER identified 117 Google dead drops, where data on Google web apps is accessed via unique file IDs. By analyzing file URLs, I traced the endusers for each file on Google Drive and Docs, finding both were hosted by the same user. VADER also discovered dead drops on Dropbox and GitHub, with 11 and seven malware instances, respectively. One WordPress dead drop and its de-manipulation recipe were included in my findings, along with three more WordPress dead drops uncovered through proactive discovery (see §3.2).

Row *Total*, Column 4 reveals spikes in DDR adoption from 2012–2022, showing a cycle of growth, delayed remediation, and resurgence. This trend, especially the recent

rise, highlights the need for proactive discovery to curb the reemergence of DDR malware, particularly as new strains appear.

In total, VADER identified 8,906 samples and 200³, including 1,199 with backups. VADER also uncovered seven unique de-manipulation recipes, demonstrating the scalability of my approach. I am collaborating with web app providers on remediation and have disrupted at least 6,674 DDR samples by removing their dead drops.

3.4.2 VADER-Enabled Proactive Discovery

The rise of DDR malware underscores the need for proactive measures to uncover hidden dead drops. Using de-manipulation recipes from DDR malware samples, I applied them to the Netresec [112] network traffic dataset, summarized in Table 3.4. Column 1 lists the recipes used to decode web app responses. After decoding, I cross-referenced the data with URLhaus [137] and VirusTotal [31] to identify potential C&C addresses, recording infected web apps and dead drops in Columns 2 and 3. Column 4 shows unique C&C addresses, while Column 5 provides the first submission date (FSD) from URLhaus, correlating with the C&C server discovery date. Column 6 reports active C&C addresses. To determine C&C server liveness, domain-based addresses were checked for website availability, and IP-based addresses were verified using `nmap` to confirm open ports, following a less intrusive method from prior research [32].

To evaluate VADER in real-world scenarios, I assessed false positives (FPs) and false negatives (FNs), as shown in Columns 7 and 8. An FP occurs when a dead drop candidate is later found to serve benign purposes. Since many web apps (e.g., OneDrive, Dropbox, Discord) do not disclose user activity details (e.g., login times, IP addresses), determining intent behind a dead drop is difficult. Therefore, I rely on service providers for verification, marking a dead drop as an FP if it remains active after notification. This helps estimate the upper bound of FPs. FNs occur when a dead drop was created before its C&C address

³Analyzing 100k malware samples revealed 273 dead drops (see §B.4).

Table 3.4: Applying Recipes On Network Traffic Dataset To Proactively Discover Dead Drops

Recipe	Web Apps	# Dead Drops	# Unique C&C	FSD ¹	Online ²	FP ³	FN ⁴
Str Prs + Base64	Pastebin	1	1	2023-12-14	1	0	0
	Retry.co	1	1	2024-02-08	1	0	0
	GitHub	1	1	2023-12-22	1	0	0
	Privatebin	1	1	2024-02-08	1	0	N/A
	OneDrive	3	3	2023-11-14	1	0	N/A
	Discord	2	2	2023-12-13	1	0	N/A
	Dropbox	1	1	2023-11-17	1	0	0
Base64	Pastebin	10	9	2024-02-01	5	2	3
	Kpaste	7	7	2018-11-27	4	0	N/A
	GitHub	1	1	2024-01-31	0	0	0
	Dropbox	2	2	2023-12-21	1	0	0
	Discord	6	6	2019-05-17	5	0	N/A
	OneDrive	3	3	2023-06-08	2	0	N/A
Base16 + Chr Rot	Pastebin	1	1	2023-12-21	1	0	0
	OneDrive	1	1	2023-10-23	1	0	N/A
	Discord	2	2	2021-04-29	1	0	N/A
XOR + Chr Rot + Base16	Pastebin	1	1	2024-01-12	1	0	0
	Justpaste.it	1	1	2023-12-12	1	0	1
	Dropbox	2	2	2024-01-24	1	0	0
	OneDrive	2	2	2023-11-15	1	0	N/A
	Telegram	1	1	2023-12-15	1	0	1
XOR + Chr Rot	Codepad	2	2	2024-02-19	1	0	N/A
	Retry.co	1	1	2024-02-13	0	0	1
	Dropbox	2	2	2023-06-15	1	0	0
	OneDrive	1	1	2024-01-25	0	0	N/A
	Discord	2	2	2023-11-26	2	0	N/A
Str Prs	OneDrive	3	3	2023-11-27	2	1	N/A
	Telegram	1	1	2024-01-29	1	0	1
	Discord	3	3	2024-01-09	2	0	N/A
	Pastebin	7	3	2022-10-13	2	1	1
Total	11	72	67	2018-11-27	43	4	8

1: First Submission Date on URLhaus [137]. 2: Number of live C&C servers.

3: A false positive (FP) is reported if a proactively identified dead drop is not taken down by web app providers.

4: A false negative (FN) is reported if the creation time of the dead drop is earlier than the time the C&C address is added to the blocklist.

was added to the blocklist, reflecting VADER’s ability to detect dead drops quickly. For platforms that do not disclose content creation times (e.g., OneDrive), these entries are marked N/A in Column 8.

From Table 3.3, VADER identified seven web apps hosting dead drops. Recipe-enabled proactive discovery, shown in column 2 of Table 3.4, revealed 11 web applications, an increase of 57. 1%. The use of de-manipulation recipes identified 72 dead drops, averaging 6.5 per web app (Column 3), highlighting the significant role of recipe identification in

Table 3.5: De-Manipulation Recipes Identified.

De-manipulation Recipes	Temporal Changes (2012-2022)	#M	#F	Web App	#A	Content	Recipe Complexity (# of basic blocks)		
							Min	Avg	Max
Single Algorithm									
Base64		964	8	Pastebin	8	Paste	26	34	43
Str Prs		1,788	4	Pastebin	6	Paste	13	19	27
		9	1	Google Drive	1	File	4	4	4
		137	8	Google User Cont.	115	File	3	7	10
		9	1	Google Drive	1	File	4	4	4
		11	2	Dropbox	2	File	5	7	8
Combination of 2 Algorithms									
Str Prs + Base64		3,301	26	Pastebin	32	Paste	32	41	47
		335	8	X	8	Post	37	37	37
XOR + Chr Rot		138	3	X	3	Post	6	8	11
Base16 + Chr Rot		4	1	WordPress	1	Blog post	17	17	17
Combination of 3 Algorithms									
Base16 + Str Prs + Str->Int		362	13	Blockchain	24	Txn ID	42	47	53
		1,838	68		14	Wallet ID	53	61	69
XOR + Chr Rot + Base16		7	2	Github	2	Repository	17	18	23
Total (≈ 1.5 algorithms/malware)		8,906	110 ¹		200 ¹	Avg.	20	23	27

: This is not the total column sum but the total unique categories.

1: This is not the total column sum but the total unique categories.

uncovering threats. Notably, only two recipes involved a single de-manipulation layer (Base64 and String Parsing). With over 40% of dead drops using multiple layers, brute-force approaches are infeasible, emphasizing VADER’s effectiveness.

Next, Column *FSD* lists the submission dates of C&C addresses behind dead drops, with the earliest from six years ago, showing the persistence of DDR malware. Comparing Column 4 and Column 6, over 64% of C&C addresses remain active, highlighting VADER’s significant contribution.

In Column 7, VADER achieved a low FP rate of 5.56% among the 72 identified dead drops. All FPs originated from single-layer recipes, likely due to benign uses of encoded IPs and URLs. With over 40% of dead drops using multiple techniques, VADER proves effective in uncovering hidden threats. Additionally, eight FNs were observed, reflecting the limitations in identifying dead drops on their creation day, but these will be mitigated by blocklist updates.

Overall, VADER achieves a recall rate of about 90% and an F1 score of 92%, demonstrating its effectiveness in identifying and mitigating DDR malware threats. Moreover, I have removed 94.4% of identified dead drops through collaboration with web app providers.

3.5 Pre-Deployment Evaluation

VADER is built upon S2E [38] for concolic analysis, with custom code for localizing DDR logic and identifying de-manipulation recipes. It integrates AVClass2, a widely utilized malware labeling tool [1], [32], [55], [140], [141], [142].

3.5.1 DDR Logic Localization

To evaluate VADER’s ability to identify DDR logic, I tested it with five known DDR malware families, each with five randomly selected variants, totaling 25 ground truth samples. The performance evaluation is summarized in Table 3.6, which lists the web app, domain,

malware family, and relevant reports detailing the ground truth, including the web app connection. Accuracy metrics such as true positive (TP), false positive (FP), and false negative (FN) values are also provided. Overall, VADER successfully localized the DDR logic in 23 out of 25 samples, achieving a 92% recall rate and an F1 score of 96%.

In my analysis, I identified two false FNs in *msil* (Table 3.6, Row 5). Manual review confirmed DDR malware, but VADER missed the dead drop link due to unresolved symbolic constraints, though this was rare. Additionally, VADER can analyze a DDR malware sample even when associated dead drops are inactive. For instance, in analyzing *connie* (Row 2 of Table 3.6), the dead drop was inactive, yet VADER used concolic analysis to successfully localize the DDR logic. With only two FNs and a 92% recall in DDR logic localization, I am confident in VADER’s ability to identify DDR malware and extract de-manipulation recipes for proactive dead drop discovery.

3.5.2 Validating Symbolic Expression Matching

Upon confirming that VADER can locate DDR logic with high accuracy, I now need to validate whether VADER can detect the decoding algorithm regardless of how it is implemented in the malware. This requires me to: (1) compare the source code similarity across all decoding algorithm implementations to ensure they are distinct (i.e., XOR must not match with Character Rotation), and (2) compare the equivalence of symbolic expressions for each pair of algorithms to demonstrate that de-manipulation algorithms in the same class consistently produce high-confidence matches, regardless of implementation details (i.e., BASE64 source code implementations).

Source Code Similarity. I selected up to three implementations of each algorithm listed in Table 3.2 from open-source repositories or libraries. Importantly, VADER is flexible and can be extended to support additional algorithm implementations. Using Moss [143], a widely used software similarity framework, I compared all implementations and found a 0%

Table 3.6: Validating DDR Logic Localization.

Web App	Domain	Family	Source	#	TP	FP	FN
Blockchain	blockchain.info	pony	[106]	5	5	0	0
Google Drive	drive.google.com	doina	[103]	5	5	0	0
Pastebin	kryptik	pastebin.com	[1]	5	5	0	0
Pastebin	msil	pastebin.com	[1]	5	3	0	2
X	x.com	razy	[100]	5	5	0	0
Total			25	Acc. (92%)	23	0	2

match between them, confirming their distinctiveness. For instance, no XOR implementation matches another, nor does it resemble implementations from other algorithm classes. Detailed results are provided in §B.2.

Symbolic Expression Matching. Given the distinct source code implementations for each decoding algorithm category, I now need to compare the equivalence of symbolic expressions generated by VADER across different implementations of decoding algorithms. The validation result is shown in Table 3.7. With a symbolic input size of 8 bytes, the expression has 256^8 (4.2 billion) possible inputs. Ideally, we would evaluate the entire input space when comparing symbolic expressions, but this is time-prohibitive. Instead, I exhaust the input space for each pair over two hours to assess the number of matches in the output space. This is sufficient, as the ratio of non-matches to matches plateaus and stabilizes after 30 minutes within the two-hour window. A graphical representation of the plateau for each set of comparisons is presented in §B.3.

As shown in Table 3.7, the symbolic expressions generated by VADER for different implementations within the same algorithm class achieve consistently high scores. This indicates that VADER can effectively recognize *functional equivalence* within algorithm classes. In contrast, expressions generated for algorithms across different classes show significantly lower scores, underscoring the differences in their operational logic. For example, Table 3.7c reveals that most comparisons ($474/554^4$) between algorithms from

⁴71/625 algorithms are compared with themselves.

Table 3.7: Validation of Symbolic Expressions Produced By VADER To Assess Functional Equivalence Across De-Manipulation Algorithm Implementations.

(a) AES - RC4.

	AES		DES		DPAPI		RC4	
	v1	v2	v1	v2	v1	v2	v1	v2
AES	v1	100 100	0	0	0	0	0	0
	v2	100 100	0	0	0	0	0	0
DES	v1	0 0 100 100	0	0	0	0	0	0
	v2	0 0 100 100	0	0	0	0	0	0
DPAPI	v1	0 0 0 0 100 100	0	0	0	0	0	0
	v2	0 0 0 0 100 100	0	0	0	0	0	0
RC4	v1	0 0 0 0 0 0 0 100 100	0	0	0	0	0 100 100	0
	v2	0 0 0 0 0 0 0 0 100 100	0	0	0	0	0 100 100	0

(b) XOR - String Parsing (Str Prs).

	XOR			Chr Sub			S $\leftrightarrow$ I		Chr Rot			Str Prs	
	v1	v2	v3	v1	v2	v3	$\rightarrow$	$\leftarrow$	v1	v2	v3	v1	v2
XOR	v1	100 100 100	11	0	0	11	0	0	0	0	0	0	0
	v2	100 100 100	0	0	0	11	0	0	0	0	0	0	0
	v3	100 100 100	0	0	0	11	0	0	0	0	0	0	0
Chr Sub	v1	0 0 0 100	92	94	0	0	0	0	0	0	0	0	0
	v2	0 0 0 86	100	84	11	0	0	0	0	0	0	0	0
	v3	0 0 0 92	91	100	11	0	0	0	0	0	0	0	0
Str, Int	$\rightarrow$	11	11	11	1	0	0 100	4	0	0	0	0	0
	$\leftarrow$	0	0	0	0	0	0 22 100	0	0	0	0	0	0
Chr Rot	v1	6 4 6	2	6	0	0	0 100	100 100	100	0	0	0	0
	v2	0 0 0	0	0	0	1	0 100	100 100	100	0	0	0	0
	v3	0 0 0	0	0	0	1	0 100	100 100	100	0	0	0	0
Str Prs	v1	0 0 0	1	0	0	11	0	13	3	3	100	99	99
	v2	0 0 0	2	0	0	11	0	19	2	3	99	100	100

(c) Base Decoders (16, 32, 64, 85).

	Base16			Base32			Base64			Base85		
	v1	v2	v3	v1	v2	v3	v1	v2	v3	v1	v2	v3
B16	v1	100 98 92	16 19 11	28 0 16	21 9 42							
	v2	89 100 72	15 8 23	50 51 52	32 14 7							
	v3	96 91 100	14 23 22	12 9 27	11 24 22							
B32	v1	12 11 14	100 97 98	62 68 64	1 22 13							
	v2	1 1 13	98 100 97	2 12 9	0 17 8							
	v3	13 21 16	92 83 100	3 9 22	13 4 3							
B64	v1	70 14 2	0 0 27	100 44 77	19 0 0							
	v2	1 16 4	3 4 4	97 100 100	21 14 5							
	v3	4 40 16	11 3 21	89 100 100	7 18 10							
B85	v1	17 45 46	23 42 62	41 37 19	100 91 93							
	v2	10 61 42	29 37 58	38 52 43	87 100 92							
	v3	32 15 29	22 12 33	19 27 20	89 92 100							

different classes result in a 0% similarity match.

A specific case is observed in Table 3.7c, Row 7, where the symbolic expression for Base64 v1 shows low similarity scores when compared to those of other versions. Upon investigation, I determined that the match rates of 44% and 77% for Base64 v2 and v3, respectively, are attributable to extensive error-checking routines present in Base64 v1. These routines cause VADER to fork into two distinct paths during symbolic execution: (1) a success path and (2) a failure path that returns null. In contrast, Base64 v2 and v3 employ less stringent error-checking mechanisms, which result in symbolic expressions that predominantly reflect successful operations. This difference explains the observed lower similarity scores when v1 is compared to v2 and v3.

When comparing Base64 v2 and v3 to v1, I observed match rates of 97%, 89%, and 100%, respectively (Table 3.7c, Rows 8 and 9, Column 7). This is because v2 and v3 lack the extensive error-checking routines of v1. Since all versions handle 8-byte input, if v2 executes symbolically first, it guides VADER along v1's success path, resulting in higher match rates. This shows how differences in error-checking impact symbolic exploration and similarity metrics.

Base decoding classes (e.g., Base16, Base32, Base64) all use lookup tables for character translation, creating a baseline similarity due to shared structural elements. As a result, only six out of 144 comparisons had a 0% match, far fewer than the expected 108 for unrelated Base classes. However, the similarity noise for cross-class comparisons remains lower than in same-class comparisons. This ensures the observed similarities are not sufficient to determine functional equivalence between unrelated Base algorithms.

Finally, my analysis shows that VADER can identify de-manipulation algorithms across different implementations. For optimal large-scale performance, I select algorithm implementations with in-class match rates over 90% and cross-class rates under 25% (e.g., v1). This selection minimizes false positives and improves VADER's reliability in identifying de-manipulation recipes.

Table 3.8: De-Manipulation Recipe Identification.

Web App	Family	Source #		Algorithm	TP	FP	FN
Blockchain	pony	[106]	5	String Parsing, Base16	5	0	0
Google Drive	doina	[103]	5	None	0	5 ¹	0
Pastebin	kryptik	[1]	5	Base64	5	0	0
Pastebin	msil	[1]	5	Base64	5	0	0
X	razy	[100]	5	String Parsing, Base64	5	0	0
None	spora	[124]	5	AES	4	0	1
Total			30	Acc. (83%)	24	5	1

1: FPs are caused by plaintext IP address format validation, which uses String Parsing. However, this has a negligible impact on proactive dead drop discovery.

3.5.3 De-Manipulation Recipe Identification

Table 3.8 shows my evaluation of VADER’s IO boundary isolation and de-manipulation algorithm verification. In all cases, the IO boundary aligned with a single, verified algorithm. Columns 1 and 2 provide details on the web app and malware family. I expanded my dataset to include *spora*, which uses cryptographic routines. The final columns of the table show the number of malware variants per family and the identified de-manipulation recipes. Overall, VADER successfully identified recipes for 30 malware samples, achieving a 96% recall rate and an F1 score of 89%.

I identified five false positives (FPs). For example, *doina* requests a plaintext IP from Google Drive, but VADER flagged it as String Parsing because its format validation resembled parsing. These FPs have minimal impact since the content is not hidden. Additionally, VADER can struggle with unresolved symbolic constraints, which limits constraint chaining for source-to-sink mapping. Despite these issues, the low FP and FN counts demonstrate VADER’s accuracy in identifying de-manipulation recipes.

3.5.4 Case Studies

My first case study demonstrates the prevalence and complexity of de-manipulation recipes. I also show that VADER effectively detects C&C addresses on Pastebin, prevalent in my

research. While my access to Pastebin is limited, this highlights the advantages of VADER, especially for web app providers with full platform access.

Prevalence and Complexity

To understand the prevalence and complexity of identified recipes in malware, I analyze the malware in this study. Table 3.5 shows de-manipulation recipes in Column 1 with their prevalence over time in Column 2. Columns 3 and 4 count malware and families, while Columns 4 and 5 list associated web apps and unique dead drops. Column 6 displays content types, and the final columns show algorithmic complexity in basic blocks.

Among the nine decoders listed in Table 3.2, VADER identified five, with XOR being the only cryptographic algorithm used by DDR malware in my dataset. This aligns with observed trends [32], [144], where encryption is often avoided due to its complexity. In contrast, decoding algorithms are platform-independent and widely supported, allowing malware authors to reuse routines across systems and environments.

From Column 3 (#M), we see that String Parsing (Str Prs) and Base64 decoding is the most common de-manipulation recipe, appearing in 3,301 DDR samples ($\approx 37\%$ of my dataset). Base64 offers obfuscation and preserves data during transport, contributing to its popularity. For instance, the *razy* malware strips five leading `) ) ) ) )` from its dead drop content and Base64-decodes the rest.

Base64 typically encodes binary data into ASCII for C&C URLs, while Base16 encodes IPs in hex to obscure them. As shown in Table 3.1, the *mudrop* malware uses Base16 decoding to extract its C&C IP from dead drop content. Other examples include Base16+Chr Rot (Row 5) and Base16+Str Prs+Str→Int (Row 6), the latter being specific to blockchain web apps. However, since the `netressec` dataset in §3.4.2 lacks blockchain connections, this technique wasn't detected.

The *Recipe Complexity* columns reflect how challenging it is to generate de-manipulation recipes. Malware using three or more algorithms may span over 69 basic blocks (Row 6).

Even simpler cases, like Str Prs, which extracts and parses dead drop content, can reach up to 27 basic blocks. On average, each sample uses 1.5 de-manipulation algorithms per recipe. Despite this, VADER effectively generates these recipes. I have shown how they enable discovering dead drops from network traffic and will next demonstrate their use in analyzing web app content.

Table 3.9: VADER’s Banload Analysis.

VADER's Banload Analysis Output		
Web App Account	http://pastebin.com/raw/jYzmPCEr	
↳Dead Drop Content	c2t5cGUubXlkZG5zLm1l	
↳Recipe	Base64 Decoding	
↳C&C Address	skype.myddns.me	
Proactive Discovery: Pastebin		
Dead Drop	Content	C&C Address
0hx614sm	c2t5cGUubXlkZG5zLm1l	skype.myddns.me
4yAyQVev	d3d3LnRvcDE2OC5vcmc=	www.top168.org
AHtqg5QQ	c2t5cGUubXlkZG5zLm1l	skype.myddns.me
FaZqCHMN	c2t5cGUubXlkZG5zLm1l	skype.myddns.me
grHusy2R	c2t5cGUubXlkZG5zLm1l	skype.myddns.me
L2QEYA3P	d3d3LnRvcDE2OC5vcmc=	www.top168.org
V7Cm4Vid	c2t5cGUubXlkZG5zLm1l	skype.myddns.me
Vr8Bequa	c2t5cGUubXlkZG5zLm1l	skype.myddns.me
XcHYKsYP	c2t5cGUubXlkZG5zLm1l	skype.myddns.me
WpU6vm5L	cHJvZmFzc2lzdGFuY2UuY29t	profassistance.com
xj8s4nF1	MTIxLjIyNi4yMzcuNQ==	121.226.237.5
EALLXQZw	MTEwLjE4Mi43OC4xNg==	103.254.75.18
DdEhn3D7	ZnVlbHJlc2N1ZS5pZQ==	fuelrescue.ie

Proactive Discovery Via Web App Scanning

Banload uses Pastebin at <http://pastebin.com/raw/jYzmPCEr> as its dead drop. VADER identified `c2t5cGUubXlkZG5zLm1l` and the Base64 de-manipulation recipe (Rows 1 and 2 in Table 3.9). Applying this recipe, VADER decoded the C&C address `skype.myddns.me` and can use this de-manipulation recipe to proactively discover previously undisclosed dead drops.

With VADER, data is randomly extracted from Pastebin using eight-character labels, de-manipulated via the recipe, and parsed with regex for IPs or URLs. These addresses are checked against URLhaus or VirusTotal for malicious activity, and associated accounts are

flagged for remediation. As noted in §3.2.3, web app providers can scan platforms at scale for abuse, demonstrating VADER’s potential for proactive dead drop discovery.

In Table 3.9, I present previously undisclosed dead drops and C&C addresses. Column 1 lists the discovered Pastebin dead drops, Column 2 shows the dead drop content, and Column 3 provides the associated C&C addresses. My proactive approach uncovered 13 previously unknown dead drops, with seven linked to the same C&C address as *banload*. Of the six remaining C&C addresses, five were unique. These results highlight that, even with limited data extraction resources compared to web app providers, VADER combined with internal tools can already enhance platform security and deliver proactive outcomes.

3.6 Discussion

Brute-Forcing Efforts. Web app providers can take content from their platform to use as input to a DDR malware sample and observe possible C&C connections. Instead, VADER enables providers to analyze malware once, extract its de-manipulation recipe, and apply it at scale to identify dead drops efficiently. While brute-forcing known de-manipulation algorithms may appear straightforward, multi-layered de-manipulation and the infinite parameter space required by some algorithms render this approach infeasible. VADER overcomes these challenges by using symbolic constraints to automatically recover de-manipulation logic, regardless of layers or parameters.

Evasion Techniques. VADER addresses many malware evasion techniques by hooking system APIs (see §3.3.1). Although comprehensive evasion mitigation is out of scope, VADER models commonly observed evasive APIs identified by industry and academic experts [145], [146], [147], [148]. For example, when GetLocaleInfo is invoked to detect a bot’s location, VADER injects symbolic return values, tricking the DDR malware sample into continued execution. This enables effective analysis even with evasive behavior. My implementation, evaluated on a malware sample set collected between 2012 and 2022,

achieves 92% accuracy in localizing the DDR logic. Thanks to VADER’s extensible design, web app providers can add support for emerging evasion techniques. A full list of modeled evasive APIs is in §B.1.

Advanced Packers and Obfuscators. Advanced packers (e.g., VMProtect) or obfuscators (e.g., O-LLVM) can complicate DDR malware detection and recipe extraction for web app providers. These challenges are not unique to VADER but reflect broader issues in symbolic analysis, where handling such techniques often requires added human effort. Several approaches have been proposed [99], [149], [150], [151] to recover the semantics of obfuscated code. As VADER does not rely on any special functionality in S2E, web app providers can readily extend it to support advanced packers and obfuscators in future DDR malware.

Generalizability. Expanding VADER is straightforward; new de-manipulation algorithms can be added by integrating their source code. While my dataset lacks fully customized de-manipulation algorithms, such extensions are feasible. In these cases, the recipes generated by VADER would model the algorithms through character operations. I recognize the potential challenges fully customized algorithms may pose to S2E. However, as discussed earlier, VADER’s methodology is tool-agnostic, ensuring compatibility with more advanced concolic analysis frameworks.

3.7 Related Work

Malware-Driven Web Application Abuse. Existing research has investigated web apps abused for botnet activities, focusing on end-user perspectives [19], [21], [53], [152], [153], [154], [155], [156]. Recent efforts have examined the intricate relationship between web apps and the malware that leverages them for victimization [1]. While studies have targeted specific malware families [101], [102], [106], [129] and blockchain abuse [3], [104], [105],

[111], [157], their findings largely propose reactive countermeasures. Before VADER, proactively discovering dead drops remained unaddressed.

Data Manipulation Identification. Previous research focused on the extraction of binary code using dynamic or symbolic analysis [150], [158], [159], [160], [161], [162], [163], [164], [165], [166]. These methods are limited by code coverage or reliance on available network endpoints. In contrast, VADER’s concolic analysis improves malware exploration even with dead endpoints. While ByteWeight [167] uses deep neural networks to identify de-manipulation algorithms, it requires a large training dataset. VADER, however, uses symbolic expression matching to identify de-manipulation algorithms regardless of implementation.

Leveraging Concolic Analysis. Concolic analysis has been widely used by the research community in bug identification [72], [168], [169], test case generation [170], [171], [172], dynamic analysis [63], [173], [174], [175], [176], and taint analysis [177], [178], [179], [180]. However, since these related works are largely geared towards vulnerability analysis, they are not easily extensible to help web app providers mitigate DDR practices. In contrast, VADER is among the first proposals to help web app providers confirm DDR capability from malware samples and extract de-manipulation recipes. This capability enables web app providers to proactively identify dead drops.

C&C Address Identification. Existing techniques detect C&C addresses by analyzing malicious network traffic [89], [93], [94], [96], [97], [98], but these typically identify C&C addresses for individual malware samples. In contrast, VADER identifies multiple dead drops from a single malware sample, which may conceal different C&C addresses. While tools like Cyberprobe [95] and ExecScent [107] probe remote endpoints to detect C&C addresses, they’re ineffective against DDR malware that hides C&C addresses behind dead drops. Instead, VADER uncovers previously undisclosed dead drops, enabling web app

providers to address abuse and counteract botnets.

3.8 Conclusion

This paper presents VADER, a framework for proactively uncovering and neutralizing dead drops used by DDR malware by dynamically resolving C&C addresses on popular web apps. Analyzing 100k malware samples, VADER identified DDR malware from 110 families and 273 dead drops across seven web apps. Using de-manipulation recipes from DDR malware samples, VADER improved dead drop identification by 57.1% in network traffic. Case studies revealed that over 40% of samples use advanced de-manipulation techniques. VADER removed 94.4% of identified dead drops, disrupting 6,674 DDR malware samples. This work shows the potential for web app providers to adopt VADER for early detection and proactive dead drop remediation.

CHAPTER 4

PULLING THE STRINGS: PROACTIVE FORENSICS ON BLOCKCHAIN FRAUD

Abstract

Criminals, using crypto wallets referred to as Deceptive Creator Wallets (DCWs), have orchestrated fraudulent activities by luring victims to transfer funds to fraud smart contracts. Since it is almost impossible to reverse the transactions or pinpoint the true identity of the criminals, the industry has turned to flagging such contracts as user warnings. However, current mitigation efforts focus on individual contracts, overlooking the DCWs behind the scenes. Consequently, my research found that this oversight allows fraud to thrive. To address this, I developed CoCo, an automated forensic analysis pipeline that processes a single fraud contract and generates evidence that the legal authorities need to mitigate the fraud. Applying CoCo to 157 confirmed fraud contracts, my research uncovered 1,283,198 associated contracts linked to 91 DCWs, responsible for 2,638,752 ETH (\$2,089,504,682) in illicit profits. More alarmingly, CoCo traces the fraudulent activities back to September 2017. In response, I am closely collaborating with Etherscan and the FBI to combat the fraud identified in my study.

4.1 Introduction

Smart contracts have been exploited to orchestrate fraudulent activities, resulting in financial losses [181], [182]. In this scenario, the criminals use their crypto wallets, referred to as Deceptive Creator Wallets (DCWs), to deploy fraud smart contracts. The criminals then lure victims with false promises, leading them to transfer funds to these contracts, which then illicitly divert the victims' assets to criminal-designated *recipients*. The irreversible and anonymous nature of the blockchain makes it impossible to reverse the transactions or

to identify these criminals. To mitigate this, the industry flags fraud contracts as warnings to users, as illustrated in Figure 4.1 on Etherscan [183]. In fact, several works have been proposed to automate fraud contract reporting [184], [185], [186], [187], [188], [189], [190], [191], [192], [193], [194], [195], [196], [197], [198]. Unfortunately, current mitigation efforts focus on individual contracts, overlooking the DCW orchestrating the fraud.

Compounding the issue is the low cost of smart contract deployment, as DCWs can continuously deploy new fraud contracts to avoid detection. Additionally, the blockchain not only enables wallets to deploy contracts but also allows contracts to deploy others, giving DCWs a fast-track method for fraud contract creation through continuous invocation. DCWs use this to conceal the recipients of fraud contracts by *dynamically resolving* on-chain data. This approach leads to the quick deployment of complex, interconnected fraud networks, where these *associated contracts* exhibit various *capabilities*, including asset transfer and new contract deployment.

Imagine this forensic scenario: FBI agents receive a report concerning a contract suspected of being involved in fraudulent activities. Ideally, FBI agents would go beyond merely mitigating the reported fraud contract; they would proactively delve into investigating the DCW orchestrating these activities to collect critical evidence. This effort aligns with the principles of the Uniform Electronic Transactions Act (UETA) [199] and National Commerce Act (E-Sign Act) [200], which recognizes the importance of *electronic agents* (DCW in this paper) in the initiation, execution, and management of electronic contracts. Motivated by the UETA and E-Sign Act, this evidence should encompass: ❶ The associated contracts from the same DCW along with their recipients. This evidence reflects the concept of *managing electronic contracts*, as defined by the UETA and E-Sign Act. ❷ The provenance of dynamically resolved recipients, related to the act of *initiating electronic contracts* as delineated in the UETA and E-Sign Act. ❸ The attribution of capabilities to the contracts. This evidence is motivated by the act of *executing electronic contracts*, as highlighted in the UETA and E-Sign Act. FBI agents could then submit

collected evidence to the court. Upon authorization, FBI agents could utilize Evidence ❶ to flag additional accounts (i.e., wallets, contracts) on the blockchain and freeze assets [201], effectively disrupting the fraudulent activities. Additionally, Evidence ❷ sheds light on the *origins* of dynamically resolved recipients, providing agents with indicators of early recipient changes and enabling more proactive mitigation. Finally, the analysis of contract capabilities in Evidence ❸ uncovers targeted mitigation strategies for specific capabilities (e.g., monitoring contracts designating fraud contract recipients).

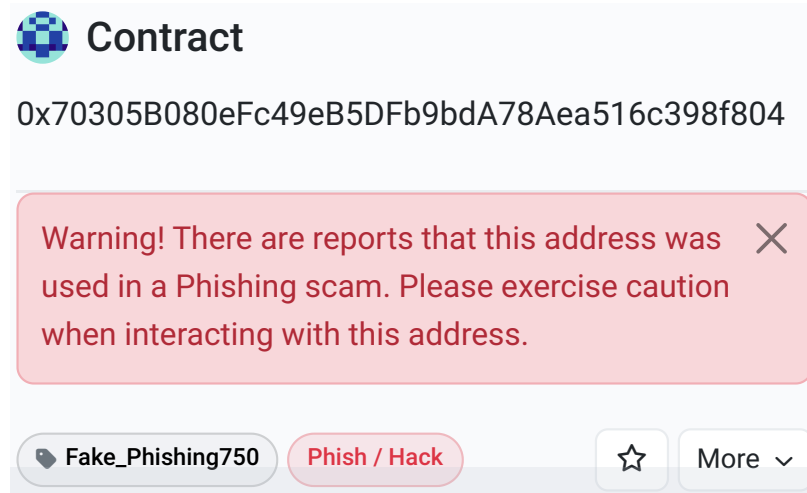


Figure 4.1: Flagged fraud contract.

Traditionally, FBI agents would acquire such evidence by relying on *explicit* clues, the historical transactions. In fact, prior research [184], [185] leveraged these explicit clues to *identify* fraud contracts. However, these contracts are just the starting point for forensic analysis. A smart contract can execute various transactions depending on different conditions. Consequently, the FBI agents quickly discover that a forensic approach that focuses strictly on *explicit* clues might cause investigators to omit evidence from yet-to-be-executed transactions, thereby halting the investigation until the relevant transactions occur.

From a different perspective, contract transactions on the blockchain are direct results of their implementations. Therefore, I shifted my focus to how a smart contract's implementation could reveal *implicit* clues of fraudulent activities orchestrated by the DCW,

such as (1) future transactions that the DCW’s contract is programmed to execute, (2) the origin of the recipients used by the DCW’s contract, and (3) the capabilities with which the contract is equipped. In fact, several works have been proposed [202], [203], [204], [205], [206], [207], [208] to perform program analysis on smart contracts. However, these techniques target vulnerability detection in benign contracts. The significant distinction between identifying vulnerabilities and discerning fraud capabilities means that these methods are not readily adaptable to extract forensic clues from fraud contracts and the DCW operating behind the scenes.

Drawing inspiration from real-world forensic investigations that gather evidence from various clues at crime scenes, I propose combining explicit clues with advanced program analysis of fraud contracts for enhanced forensic analysis. Based on this insight, I developed CoCo¹, a post-detection forensics pipeline enabling FBI agents to extract three key pieces of evidence of the DCW. Given one fraud contract, CoCo first uses *Associated Contracts Recovery* (§4.3.1) to pinpoint associated contracts by utilizing explicit clues. This process mirrors the method of unraveling a criminal network, starting from one identified suspect and extending to the ringleader, additional suspects, and the middleman orchestrating the recruitment. Next, CoCo conducts symbolic analysis on each contract deployed by the DCW (§28), identifying all potential recipients without depending on historical transactions. This step is comparable to identifying all possible contacts of the suspects, even those yet to engage in criminal activity. CoCo then combines explicit and implicit clues in *Recipient Provenance Investigation* (§4.3.2) to uncover the *origin* of dynamically resolved recipients. This resembles discovering the middleman’s hidden list of recruits, enabling authorities to detect future suspect recruitment early. Finally, CoCo applies *Capability Attribution Analysis* (§4.3.3) to assign specific capabilities to each contract deployed by the DCW, similar to a detective deducing the role each suspect plays in a criminal network to tailor countermeasures for each role.

¹CoCo: ForensiCs on CreatOr Wallet Behind Smart COntract Fraud

Deploying CoCo on 157 confirmed fraud contracts, my work identified 1,283,198 associated contracts across 91 DCWs, making a total of 2,638,752 ETH (\$2,089,504,682) illicit profit. More alarmingly, CoCo’s analysis tracked the fraudulent activities back to September 2017. I am closely collaborating with Etherscan [183] and the FBI [209] (§C.2) to mitigate the fraud uncovered by my study. Lastly, I have made CoCo available at <https://github.com/CyFI-Lab-Public/COCO>.

4.2 Motivation

Accounts (i.e., wallets and contracts) on Ethereum are identified by 40-character hexadecimal strings, known as addresses. Similarly, transactions on Ethereum can be identified by 64-character hexadecimal strings, commonly referred to as transaction hashes. To enhance the readability of this paper, I have used abbreviations of the last six characters of the addresses and transaction hashes with the prefixes W- for wallet, C- for contract, and T- for transaction. For example, the fraud contract shown in Figure 4.2a can be identified by address, 0x70305b080efc49eb5dfb9bda78aea516c3 98f804. In this paper, I will refer to it as C- 98f804. For the full addresses and transaction hashes used throughout this paper, readers are directed to Table C.1 in §C.3.

4.2.1 Background

Smart Contract. A smart contract is a self-executing program operating on a blockchain, set to action when specific predefined conditions are satisfied. The execution of a smart contract comprises three core components: (1) the contract’s bytecode, which contains the compiled instructions of the smart contract code, (2) an execution context, which includes the Program Counter (PC), available gas, stack, and memory, These components manage the execution flow and intermediate computations during the execution of a smart contract, (3) a persistent storage mechanism specific to each smart contract, providing a mapping from 256-bit words to corresponding 256-bit words. This storage is utilized by smart contracts to

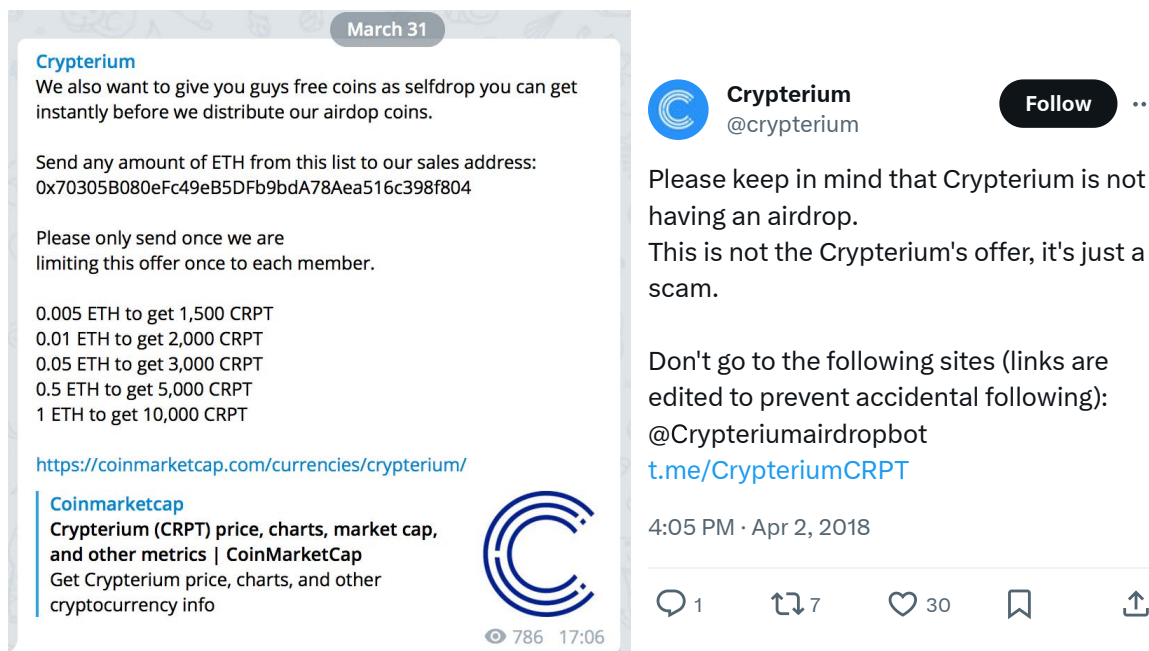
preserve the state across transactions.

Transaction & Trace. A transaction represents an action initiated by an external account (i.e., a user's wallet) that interacts with the blockchain. Transactions can encompass various actions, such as transferring ETH, interacting with a smart contract, or even deploying a new contract. Traces, on the other hand, provide step-by-step execution logs that detail all the internal calls and state changes triggered by a transaction. For example, consider a transaction in which a user sends ETH to a smart contract, which then distributes this ETH to other addresses based on its logic. The transaction itself records the user's action of sending ETH to the contract. However, the trace of this transaction would reveal the detailed sequence of events inside the contract, such as the contract calling its internal functions to distribute ETH to other addresses.

Event Logs. Event logs are small data amounts on the blockchain that utilize five opcodes (LOG0 to LOG4) for log emission. Each log can include up to four 32-byte topics and a data section. The topics typically describe the event, often incorporating the event signature—a Keccak-256 [210] hash of the event name along with its parameter types. This allows for targeted searches, such as identifying logs for specific events or addresses. The data section complements the topics by providing non-searchable additional information. It can contain more complex details, like arrays or strings, making the logs both comprehensive and flexible. Consider an event like `event Event(address indexed from, address to)`. For this event, the first topic is derived using a Keccak-256 hash on the event signature `Event(address,address)`. The second topic is the indexed `from` address. Since the `to` parameter is not indexed, its value is stored in the log's data section.

4.2.2 Preliminary Forensics Investigation

To illustrate the method of deriving the three pieces of evidence, I devised a hypothetical forensic investigation scenario modeled on a real-world case. FBI agents received



(a) Fraud message on Telegram.

(b) Confirmation on X.com.

Figure 4.2: Fraud contract in the real world.

information from an individual who reported being defrauded by a deceptive message on Telegram [14]. As shown in Figure 4.2a, the DCW propagated a deceptive message impersonating Crypterium [211], a blockchain startup. In this message, the DCW promised a giveaway, conditional upon interactions with the contract identified by C-98f804. Upon delving deeper, the agents uncovered several similar fraudulent messages disseminated across various platforms, as listed in Figure C.1 and Figure C.2 in §C.1. To ascertain the fraudulent nature of this contract, the agents located a confirmation from Crypterium on X.com [12], as evidenced in Figure 4.2b.

At this juncture, the agent possessed only the confirmed fraud contract address, C-98f804. To proceed with legal action, the agents required three pieces of evidence, as outlined in §4.1, which would be necessary to present in court. This would support more proactive mitigation strategies, such as flagging more fraud contracts or freezing the associated wallets or contracts (as was previously done by the FBI [201], [212]).

Associated Contracts & Recipients. Extracting Evidence ❶ consists of two steps: first, identifying the *associated* contracts and, second, pinpointing their *recipients*. My approach begins by leveraging the transparent nature of the blockchain to reveal associated contracts. Specifically, my investigation started with an analysis of transactions *to* C-98f804, leading me to transaction T-8bbae5, which deployed the contract. The sender information embedded in this transaction reveals the DCW as W-521058. Tracing back from this transaction, I further explored other contracts deployed by W-521058 directly, amounting to 395,685 contracts in total. Subsequent examination of transactions *from* these contracts unveiled a total of 251,087 contracts with historical transactions to the same recipient – W-763bc0.

Another problem that emerged during this process is that 144,597 (36.5%) contracts show no historical transactions. Sole reliance on explicit clues cannot reveal the recipients of these contracts. Instead, I turned my attention to the first implicit clue: *The contract's implementation could reveal future transactions that the contracts is programmed to execute*. Despite their diverse logic, fraud contracts still need to employ CALL opcode for ETH transfers. A detailed analysis of the CALL callsites within these contracts revealed that all 144,597 contracts possess the capability to transfer victim ETH to a single recipient, W-763bc0. As shown on Row 3 of Table 4.1, this led to the generation of Evidence ❶, showing that there are 395,684 associated contracts capable of redirecting victim assets to the recipient wallet W-763bc0. Notably, one contract deployed by W-521058 remains unattributed. The subsequent sections introduce the methods to dissect this singular contract.

Masquerading As Inactive. The previous analysis of W-521058 shows that a constant fraud contract deployment ranged from October 2017 to January 2021. At first glance, FBI agents might conclude that the DCW ceased the fraud after January 2021 due to a halt in deploying new fraud contracts. However, they soon realized that 98.4% of transactions initiated by W-521058 after that invoked C-48d304.

Table 4.1: Recovered Evidence From COCo’s Forensics Investigation.

Fraud Contract	C-98f804
Creator	W-521058
Evidence ①	Associated Contracts: Direct: 395,685 (11) Indirect: 65,330 (0) Recipients: W-763bc0
Evidence ②	C-48d304 → Storage → Contract
Evidence ③	<i>attr</i> ₁ : Forced Transfer (395,684) <i>attr</i> ₂ : Contract Self-Replication (1) <i>attr</i> ₃ : Fraud Event Logging (395,684) <i>attr</i> ₄ : Recipient Resolution (65,330)
Fraud Impact	Victim: 232,011 ETH: 1,240,355

11: Number of contracts flagged on Etherscan [183].

This motivated me to switch my focus to contract C-48d304. My analysis of the implementation showed that C-48d304 uses opcode CREATE2 to deploy new contracts activated upon its invocation. To make things worse, the DCW used C-48d304 to indirectly deploy new contracts, which were designed to transfer victim assets. This led to the discovery of an additional 65,330 associated contracts, as detailed on Row 3 of Table 4.1, significantly expanding the scope of the investigation.

Given the newly discovered associated contracts deployed *indirectly* by the DCW, FBI agents now need to know whether these contracts share the same recipients as before. Surprisingly, FBI agents quickly realized that they could not extract recipient addresses as before by examining the address passed to the CALL opcode because the contracts dynamically resolved the recipients. This brought me to the second implicit clue: *In-depth program analysis on the contract could reveal the origin of dynamically resolved recipients.* Equipped with this implicit clue, I discovered that the newly identified contracts retrieve values from their *storage*, which are then passed as recipient addresses to the CALL opcode. Recognizing that these indirectly deployed contracts *use* storage to determine recipients, the key question became: Who *defines* this storage? To answer this question, I delve deeper into

the contract deployment chain (i.e., W-521058 $\rightarrow$ C-48d304 $\rightarrow$ *Child_Contracts*). It turned out that the parent contract C-48d304 defines the storage of the child contracts during the deployment process. Interestingly, the recipient address set by the parent contract points to the same recipient: W-763bc0. By performing the provenance analysis on the contract deployment chain, I generated Evidence ②, as shown on Row 4 of Table 4.1. The reconstructed provenance of the fraud contract deployment chain reveals that 65,330 contracts lack hardcoded recipients. Instead, these contracts determine the recipient dynamically via their own storage parameters, as defined by the parent contract C-48d304.

Capabilities Attribution. Evidence ① and Evidence ② present a good opportunity for the agents to flag the accounts mentioned therein, serving as a *reactive* strategy to mitigate the ongoing fraudulent activities. However, due to the immutable nature of the blockchain, completely eliminating the DCW from the system is almost impossible. Therefore, it becomes equally crucial for FBI agents to implement *proactive* strategies aimed at preventing future fraudulent activities. This led to my last implicit clue: *In-detailed program analysis could identify the capabilities of contracts, which can then be mapped to corresponding proactive mitigation strategies*. Specifically, as shown on Row 5 of Table 4.1, I found a total of four different capabilities presented in the associated contracts, as Evidence ③. My discovery of a hardcoded recipient in CALL led to the identification of the *Forced Transfer* capability group, prompting a *reactive* strategy to flag these accounts and their recipients. Additionally, the detection of CREATE2 opcode usage for deploying asset-transferring contracts highlighted the *Contract Self-Replication* capability group, enabling FBI agents to proactively flag future deployed contracts. Furthermore, I pinpointed 395,684 contracts with the *Fraud Event Logging* capability, using the LOG opcode to generate blockchain event logs. Given the indexability of the event logs, FBI agents can actively monitor these logs for early detection of fraudulent activities.

With Evidence ①, Evidence ②, and Evidence ③ generated, I further evaluated

the effectiveness of current mitigation efforts. Alarming, as indicated by  on Row 3 of Table 4.1, only 11 out of 461,015 associated contracts are flagged on Etherscan [183]. Compounding this issue, I analyzed historical transactions *to* these contracts, assessing the impact of fraud activities by W-521058. This led to the discovery of 232,011 unique victim addresses and an illicit profit of 1,240,355 ETH. These findings highlight the urgent need for forensic analysis focused on DCWs, and I am collaborating closely with Etherscan [183] and FBI [209] to mitigate the fraud.

Investigators With CoCo. Upon receiving a fraud smart contract report, FBI agents input the fraud smart contract address into CoCo for forensic analysis. After CoCo’s analysis, FBI agents acquire the three pieces of evidence discussed. FBI agents can then submit this evidence to the court and relate each piece of evidence to the UETA [199] and E-Sign Act [200], as demonstrated in §4.1. FBI agents could utilize Evidence ❶ to flag additional accounts involved in the fraudulent activities. Evidence ❷ provides FBI agents with the ability to detect and mark the new recipients designated by scammers to gather illicit profits even before these recipients become actively involved. With Evidence ❸, FBI agents can take appropriate mitigation actions, as detailed in §4.3.3.

4.3 Design

CoCo equips FBI agents with the techniques to investigate the DCW orchestrating the fraudulent activities using smart contracts. Starting with a fraud contract, CoCo generates Evidence ❶ by uncovering the associated contracts from the same DCW along with the recipients they could interact with (§4.3.1). Subsequently, CoCo conducts a recipient provenance analysis on dynamically resolved recipients to produce Evidence ❷ (§4.3.2). Lastly, CoCo performs *Capability Attribution Analysis* to identify and attribute capabilities to associated contracts, establishing Evidence ❸.

4.3.1 Associated Contracts Recovery

Evidence ❶ could help FBI agents freeze the assets and flag associated contracts and recipients beyond the confirmed fraud contract. To derive Evidence ❶, CoCo conducts *Associated Contracts Investigation* and *Recipients Forensic Investigation*.

Associated Contracts Investigation

Given a reported fraud contract α , CoCo begins by extracting all historical transactions directed *to* this contract, represented as $\mathcal{T} = \{t_1, t_2, \dots, t_m\}$. For instance, when considering the fraud contract C-98f804 detailed in §4.2.2, CoCo records a total of 455 transactions directed to it. Naive FBI agents might assume the first transaction txn_1 to be the creation transaction. Unfortunately, as seen in real-world situations [213] and in §4.4.2, there are instances where victims transact with a fraud contract even before its deployment, rendering the first transaction an unreliable indicator. To address this, CoCo traverses through $\mathcal{T}$ and identifies the transaction that *creates* the fraud contract, called a *creation transaction*, denoted as txn_{cr} . For the contract C-98f804, CoCo identifies $txn_{cr} = \text{T-8bbae5}$. This identified creation transaction acts as an *explicit* clue, offering FBI agents a crucial lead to determine the *sender* of the transaction as DCW's wallet, denoted as w . Ideally, the agents would then monitor all transactions initiated by the w and categorize associated contracts as $\{t_m.to | t_m \in \mathcal{T} \wedge t_m.type == CreateContract\}$.

Unfortunately, as highlighted in §4.2.1, W-521058 deployed 65,330 fraud contracts indirectly by invoking contract C-48d304. Since those transactions are a simple contract call, tracking only contract creation transactions initiated by the DCW easily could produce inaccurate Evidence ❶. This nuance underscores the importance for FBI agents to discern the *deployment chain*. To overcome this challenge, CoCo utilizes the traces of each transaction. This solution is built upon an observation: If an invoked contract deploys a new contract during a transaction, this action will be recorded in the transaction's traces, as it forms part of the subsequent activities of the invoked contract. Armed with txn_{cr} , CoCo deploys the

Algorithm 4: Identify Associated Contracts

Input: DCW w
Output: Associated Contracts Set $\mathcal{C}$

```
1  $\mathcal{C} \leftarrow \emptyset$ 
2 Function GenerateTxnGraph( $txn$ )
3    $G \leftarrow \emptyset$ ;
4   // Iterate over traces in  $txn$  to build graph
5   for  $trace \in txn$  do
6      $from \leftarrow trace.from$ 
7      $to \leftarrow trace.to$ 
8      $edge \leftarrow trace.type$ 
9      $G.add\_edge(from, to, edge = edge)$ 
10  end
11  return  $G$ 
12 end
13 // Extract all transactions From  $w$ 
14  $T_w \leftarrow GetAllTxnFrom(w)$ ;
15 // Generate contract deployment graph Of  $w$ 
16  $G \leftarrow \bigcup_{\forall txn_i \in T_w} GenerateTxnGraph(txn_i)$ 
17 // Update  $\mathcal{C}$  with contracts deployed directly
18  $\mathcal{C} \leftarrow \{edge.to \mid edge \in G \wedge edge.type == CreateContract\}$ 
19 // Identify contracts deployed indirectly
20 for  $s_i \in \mathcal{C}$  do
21    $T_{s_i} \leftarrow GetAllTxnFrom(s_i)$ ;
22   for  $trace \in GenerateTxnGraph(txn \in T_{s_i})$  do
23     // Handle different trace type
24     switch  $trace.type$  do
25       // If deploy other contracts
26       case  $CreateContract$  do
27         // Extract deployed contract
28          $s_j \leftarrow trace.to$ 
29         // Get all transactions initiated by the deployed contract
30          $T_{s_j} \leftarrow GetAllTxnFrom(s_j)$ 
31         // Track contract recursively
32          $G \cup \bigcup_{\forall txn \in T_{s_j}} GenerateTxnGraph(txn)$ 
33          $\mathcal{C} \leftarrow \mathcal{C} \cup s_j$ 
34       end
35     end
36   end
37 end
38 return  $\mathcal{C}$ 
```

methodology presented in Algorithm 4. As shown on Line 4 of Algorithm 4, for a given transaction, CoCo navigates through traces in that invocation to craft the graph $G = (\mathcal{N}, \mathcal{E})$. Here, each node $n \in \mathcal{N}$ presents an account, and every edge $e \in \mathcal{E}$ presents a trace, which could be subsequent behaviors (e.g., contract invocation, contract creation). By tracking the

trace with type *CreateContract*, CoCo is able to identify the contracts deployed indirectly.

Given the ability to identify contracts deployed both directly and indirectly from transactions, CoCo now can identify associated contracts by performing forensic analysis on the DCW's wallet w . Specifically, CoCo starts by tracking all transactions initiated by w (shown on Line 12 of Algorithm 4), designated as T_w . Then, for each transaction $txn_i \in T_w$, CoCo uses *GenerateTxnGraph* introduced on Line 2 in Algorithm 4 to distill the internal trace graph, subsequently consolidating them to form the transaction graph G for wallet w . Essentially, G encapsulates the transaction used by w to deploy contracts directly and the invocation of the contract. To delve deeper into this chain and disclose the contracts spawned by it, as shown on Line 17 of Algorithm 4, CoCo gathers traces initiated by each contract s_i , termed T_{s_i} . If a trace results in *CreateContract* (shown on Line 19 of Algorithm 4), CoCo first updates G with the transactions from this newly deployed contract recursively and then incorporates the deployment result into associated contracts.

Recipient Investigation

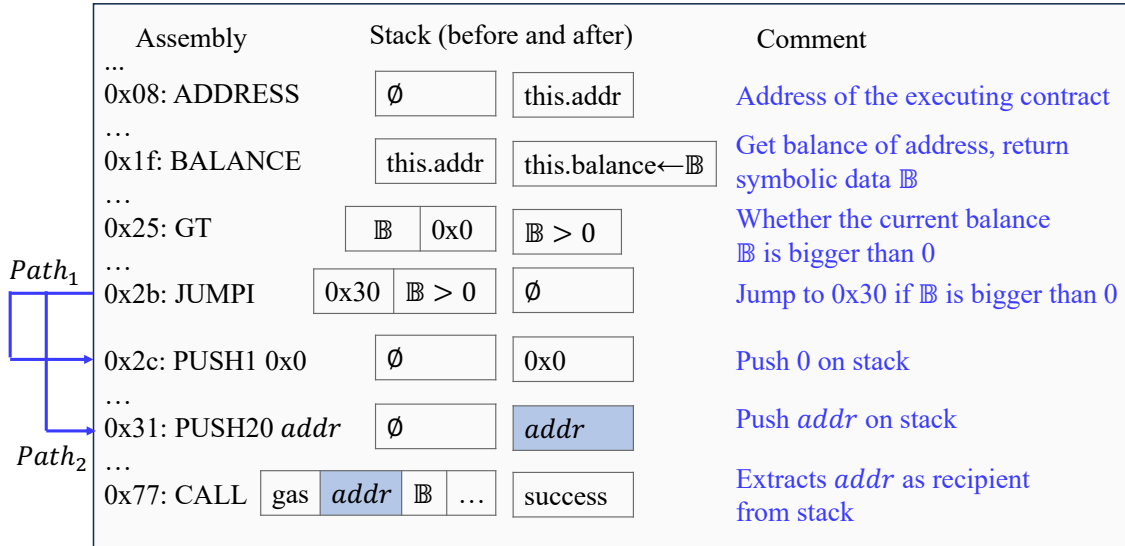


Figure 4.3: Recipient investigation of C-98f804.

The only missing piece of Evidence ❶ now is the recipients, along with the associated contracts identified. This identification enables FBI agents to flag accounts and freeze

ETH assets more proactively. However, as illustrated in §4.2.2, my initial investigation indicates that 36.5% of the fraudulent contracts deployed by W-521058 lack any historical transactions. Relying solely on, or waiting for, these explicit clues (i.e., transactions) could significantly hinder the progress of the investigation.

To overcome this challenge, CoCo conducts a transaction-agnostic symbolic analysis on each associated contract. Specifically, CoCo designates the input space $\mathbb{I}$, storage space $\mathbb{S}$, and account balance $\mathbb{B}$ as symbolic. This symbolic designation allows CoCo to conduct a multi-path exploration based on the contract’s logic. Figure 4.3 shows a segment of the opcode in fraud contract C-98f804, as discussed in §4.2.2. As shown at address 0x1f in Figure 4.3, when CoCo symbolically executes BALANCE opcode to retrieve the balance of the contract, CoCo marks the returned value as symbolic. Subsequently, this symbolic value is used by the GT opcode shown at address 0x25 to assess whether the balance exceeds 0. Upon reaching the JUMPI opcode at address 0x2b in Figure 4.3, which performs a conditional jump based on the preceding comparison, CoCo forks the execution state into two paths. One path, $Path_1$ in Figure 4.3, operates under the condition $\mathbb{B} > 0$, while the other, $Path_2$ in Figure 4.3, assumes $\mathbb{B} \leq 0$, allowing CoCo to explore behaviors in both contexts. As depicted at address 0x31 in Figure 4.3, the fraud contract deploys PUSH20 opcode to push an address $addr$ onto the stack. However, the specific purpose of $addr$ remains unclear to CoCo. Subsequently, when the CALL opcode is invoked at address 0x77 of Figure 4.3, CoCo retrieves the address $addr$ from the stack, indicating its use as a recipient address. Notably, the smart contract permits the invocation of code from other contracts via DELEGATECALL or CALLCODE. In fraud contracts, such delegation can obscure potential transaction logic in other contracts. To uncover all possible recipients used by the contract, CoCo recursively delves into contracts invoked with DELEGATECALL or CALLCODE, inheriting constraints from the caller contract. This transaction-agnostic analytical method equips FBI agents with the capabilities to analyze associated contracts and pinpoint potential recipients, independent of historical transaction data.

4.3.2 Recipient Provenance Analysis

Following the derivation of Evidence ①, a subsequent challenge arises in that not all contracts employ hardcoded recipients; many use dynamic resolution. As CoCo tracks recipients from the stack during multi-path exploration (§28), dynamically resolved recipients present a challenge, lacking hardcoded addresses on the stack. This situation motivates Evidence ②, which focuses on the recipient’s provenance, including both the resolution logic and the recipient’s origin. With Evidence ②, FBI agents gain the ability to flag accounts that define recipients and monitor recipient changes at upstream, enabling proactive mitigation of fraudulent activities. CoCo employs *In Contract Resolution Analysis* and *Cross Contract Resolution Analysis* to ascertain the provenance of dynamically resolved recipients.

In Contract Resolution Analysis. CoCo determines dynamical resolution when the recipient, identified in Recipient Investigation (§28), corresponds to a symbolic value $recp_s$. Then, CoCo conducts a *backward slice* on $recp_s$ to identify the origin of $recp_s$. For instance, in analyzing the fraud contract deployed by C-48d304 (§4.2.2), CoCo uncovers that the recipient on the stack is a symbolic value. When CoCo performs backward slice on $recp_s$, it traces back to opcode AND at address 0x4d, indicating $recp_s$ is from a logical AND operation. Moving further back, CoCo encounters another AND operation at address 0x37. Continuing this trace, CoCo reaches an opcode DIV at address 0x21. This division operation implies that $recp_s$ is dependent on a division calculation. The backward slice eventually leads CoCo to address 0x1a with a SLOAD operation. At that moment, CoCo proves that $recp_s$ originates from the contract’s storage. By examining the parameter passed to SLOAD, a constant 0x00, CoCo concludes that the recipient address is located in storage location $\mathbb{S}[0]$.

Cross Contract Resolution Analysis. The previous analysis reveals only where the fraud contract loads the recipient from. However, there is no evidence of which account defines

the recipient address (i.e., the value stored in $\mathbb{S}[0]$). To address this and draw a full picture of Evidence ②, CoCo conducts Cross Contract Resolution Analysis. Notably, smart contracts rely on the CREATE or CREATE2 opcodes to deploy other contracts. Executing these opcodes results in a contract deployment transaction, and eventually, both opcodes need init code as input. Motivated by this, once CoCo determines the in-contract origin of a dynamically resolved recipient, such as $\mathbb{S}[0]$, it identifies the contract’s deployment transaction (as discussed in §4.3.1). From this transaction, CoCo extracts the init code used in the contract’s creation. It then conducts symbolic analysis on this init code to track changes to the in-contract recipient origin. For instance, in the case of associated contract C-99bcb3, CoCo pinpoints its deployment transaction with hash T-d9d53e, initiated by W-521058 via invoking C-48d304. By analyzing the direct creator contract, CoCo extracts the init code from the parameter passed to CREATE2. Subsequent symbolic analysis on this init code reveals the use of SSTORE opcode, pointing to key 0 and assigning a hardcoded address as a value. Consequently, CoCo attributes the parent contract as the origin of the recipient.

4.3.3 Capability Attribution Analysis

Evidence ③ attributes specific capabilities to contracts associated with a confirmed fraud. These capabilities guide FBI agents in implementing corresponding mitigation strategies for ongoing and future fraud prevention. CoCo uses symbolic data, introduced in §28, effectively highlighting the information flow to track the capabilities of associated contracts. An example in §4.3.2 demonstrates this: When CoCo detects symbolic data propagation from SLOAD to the recipient in a CALL opcode, it’s interpreted as *Dynamic Recipient Resolution*, triggering the *Monitor* mitigation strategy. The 10 capabilities identified by CoCo, along with semantic models and mitigation strategies, are outlined in Table 4.2. As seen on Column 4 of Table 4.2, CoCo proposes three different mitigation strategies. The function $F(\theta)$ represents flagging and freezing the account denoted by θ . $S(\theta)$ means scanning

Table 4.2: Fraud Contract Capabilities, Opcode, Semantic Models, And Mitigation Strategies.

Fraud Capability	Opcode	Semantic Models	Mitigation ¹
Forced Transfer	CALL	$\neg \text{Symb}(\text{Stack}[1]) \wedge \text{Address}(\text{Stack}[1]) \wedge \text{Stack}[2] > 0$	$F(\text{Stack}[1])$
User-Defined Transfer	CALL	$\text{Symb}(\text{Stack}[1]) \wedge \text{Stack}[1] \in \{\text{CALLDATA}, \text{MLOAD}\} \wedge \text{Address}(\text{Stack}[1]) \wedge \text{Stack}[2] > 0$	$F(\text{Stack}[1])$
Recipient Resolution	CALL	$\text{Symb}(\text{Stack}[1]) \wedge \text{Stack}[1] \in \{\text{SSLOAD}\} \wedge \text{Stack}[2] > 0$	$F(\text{Stack}[1])$ $M(\text{SSLOAD})$
Address Allocation	SSTORE	$\neg \text{Symb}(\text{Stack}[1]) \wedge \text{Address}(\text{Stack}[1]) \wedge \exists k. (\text{SLOAD}(k) == \text{Stack}[1]) \wedge (\text{CALL}(k) \vee \text{CALLDATACOPY}(k))$	$F(\text{Stack}[1])$ $M(\text{this})$
Contract Self-Replication	CREATE CREATE2	$\neg \text{Symb}(\text{Stack}[0]) \wedge \text{Code}(\text{Stack}[0])$	$F(\text{return})$ $S(\text{Stack}[0])$
Dynamic Contract Deployment	CREATE CREATE2	$\text{Symb}(\text{Stack}[0]) \wedge \text{Stack}[0] \in \{\text{SSLOAD}\} \wedge \text{Code}(\text{Stack}[0])$	$F(\text{return})$ $S(\text{Stack}[0])$ $M(\text{SSLOAD})$
Predefined Contract Redirection	DELEGATECALL CALLCODE	$\neg \text{Symb}(\text{Stack}[0]) \wedge \text{Address}(\text{Stack}[0])$	$F(\text{Stack}[0])$ $S(\text{Stack}[0])$
Dynamic Execution Delegation	DELEGATECALL CALLCODE	$\text{Symb}(\text{Stack}[0]) \wedge \text{Stack}[0] \in \{\text{SSLOAD}\}$	$F(\text{Stack}[0])$ $M(\text{this})$ $S(\text{Stack}[0])$
Address Forwarding Strategy	SSTORE	$\neg \text{Symb}(\text{Stack}[1]) \wedge \text{Address}(\text{Stack}[1]) \wedge \exists k. (\text{SLOAD}(k) == \text{Stack}[1]) \wedge (\text{DELEGATECALL}(k) \vee \text{CALLCODE}(k))$	$F(\text{Stack}[1])$ $M(\text{this})$ $S(\text{Stack}[1])$
Fraud Event Logging	LOGX	$(\text{Stack}[0] \neq 0 \wedge \text{Stack}[1] \neq 0) \vee (\text{Stack}[2], \dots, \text{Stack}[X-1]) \neq 0$	$S(\text{Stack}[2 : X-1])$ $S(\text{Mem}[\text{Stack}[0] : \text{Stack}[1]])$

¹: F flags and freezes the specified account; S scan the content across on-chain data; M monitors the designated account.

the blockchain for contracts sharing the same code as θ if θ is a contract address, or for identical data to θ if θ represents raw byte content. $M(\theta)$ indicates monitoring the account specified by θ . Notably, CoCo’s approach to symbolic analysis, independent of semantic model knowledge, requires only a one-time effort and offers ease of extension.

1. Forced Transfer. When the opcode is CALL and the first element on the stack is not symbolic (hardcoded) and the second element is greater than zero, this indicates a Forced Transfer. This capability involves transferring ETH to a hardcoded recipient address, suggesting a transfer to a specific address. Identifying and reporting these hardcoded recipients allows FBI agents to proactively flag and freeze the extracted recipient accounts.

2. User-Defined Transfer. This occurs when the CALL opcode is used with a symbolic recipient, which is specified by the user. The recipient address is usually derived from user input, typically through CALLDATA or MLOAD. Reporting this capability allows FBI agents to proactively flag the recipients specified in the user transaction.

3. Dynamical Recipient Resolution. In this scenario, the CALL opcode uses a symbolic first stack element where the recipient address is dynamically determined. This is identified when the first stack element depends on the return of SLOAD. This dynamic resolution allows the DCW to alter the recipient address. Reporting this capability allows FBI agents not only to flag the current recipient address, but also to monitor the location where the contract load recipient address, in order to detect a change in the recipient’s address before fraud occurs.

4. Address Allocation. The SSTORE opcode, coupled with a non-symbolic first stack element that is an address of an account, indicates Address Allocation capability. Here, a specific address is hardcoded into storage and subsequently used in transaction operations, which can be identified if SLOAD with a particular key equals the non-symbolic data identified previously and is followed by CALL or CALLDATA operations. Reporting this allows FBI

agents to proactively flag the account and monitor the transactions of this contract since it could update the address.

5. Contract Self-Replication. This capability is identified when either the CREATE or CREATE2 opcode is used with a non-symbolic first element on the stack. It indicates Contract Self-Replication, where a contract replicates itself using a predefined template. This is often used in fraudulent activities to propagate fraud contracts.

6. Dynamic Contract Deployment. Identified by the use of CREATE or CREATE2 opcodes with a symbolic first element on the stack, this capability suggests Dynamic Contract Deployment. It is marked by the dynamic deployment of new contracts, which can vary in nature and functionality based on the input or state, allowing for varied fraudulent activities.

7. Predefined Contract Redirection. This occurs when either the DELEGATECALL or CALLCODE opcode is used with a non-symbolic first element on the stack. It indicates a redirection of execution to another contract with a hardcoded address, typically for executing specific fraudulent actions predefined in the target contract.

8. Dynamic Execution Delegation. Identified by the use of DELEGATECALL or CALLCODE opcodes with a symbolic first element on the stack, this capability suggests the delegation of execution to different contracts based on dynamic conditions or inputs. It allows a contract to adapt its execution and fraud strategy based on real-time data or states.

9. Address Forwarding Strategy. This is observed when the SSTORE opcode is used with a non-symbolic second element on the stack that points to an account, and a key exists such that SLOAD with this key equals the value previously stored. The loaded value would be passed to DELEGATECALL or CALLCODE. It indicates a strategic storing of an address for subsequent use in delegation operations, implying a planned redirection of execution.

10. Fraud Event Logging. This is observed when CoCo encounters the LOGX opcode ($X \in [0, 4]$), which is used to log events. This capability suggests that the contract logs event that could be used to track the contract’s activities and identify fraudulent transactions.

CoCo is now equipped with the technique to produce Evidence ③ by attributing capabilities to each associated contract and proposing corresponding mitigation strategies. This categorization enables FBI agents to implement bulk mitigation actions effectively.

4.4 Case Studies

4.4.1 Case Study 1: Contracts Shared By DCWs

Deploying CoCo on the dataset surprisingly revealed contract reuse across different DCWs. Specifically, CoCo identified a total of three different bytecodes used by seven DCWs involving 16 contracts. The FBI agents may incorrectly assume that contracts sharing the same bytecode would execute the same transactions. This assumption would be valid if these contracts shared the *Forced Transfer* capability, which means that hardcoded recipients are in the code. Interestingly, CoCo reported that these 16 contracts are all equipped with the *Dynamic Recipient Resolution* capability. For example, CoCo observed that the contracts could execute SLOAD to load an address from the sixth storage slot. Then, the loaded result would be utilized as the recipient in the fraudulent transactions. By performing *Recipient Provenance Analysis*, as detailed in §4.3.2, CoCo discovered that the sixth storage slot consistently hosted the address of the *creator*. Consequently, even though these 16 contracts share three unique bytecodes, they still engaged in different transactions.

4.4.2 Case Study 2: Abuse Of CREATE2

Beyond merely scrutinizing the overarching fraudulent activities orchestrated by DCWs, deploying CoCo also enables a detailed examination of their technical evolution. During the analysis of W-521058, who has collected around 1,250,355 of illicit ETH profit, CoCo observed an evolution from manual fraud contract deployment to an automated fraud

contract deployment given C-48d304 with *Contract Self-Replication*. Compounding this issue, during the analysis of C-48d304, CoCo discovered that instead of using the CREATE opcode, the DCW employed CREATE2. The CREATE opcode computes the address of the deployed contract using Keccak256 [210] on *sender* and *nonce* where *sender* is the address of the sender and *nonce* represents the number of transactions originating from the sender's address. Although CREATE's functionality is deterministic, predicting the address of the deployed contract remains a challenge since it relies on the on-chain data, *nonce*. In contrast, instead of using *nonce*, the CREATE2 opcode facilitates the creation of contracts with a deterministic address by using *salt*, which could be arbitrarily defined in the transaction. This feature allows DCWs to predict the address of the yet-to-be-deployed fraud contract without even accessing the blockchain. Consequently, DCWs could lure victims into transacting with a 'non-existent' fraud contract [213]. At such a juncture, discerning the malicious nature of the contract becomes exceedingly challenging for the victims, as it is yet to be deployed. To substantiate this hypothesis, I observed that all 65,530 fraud contracts had victim transactions that occurred prior to the actual deployment of the fraud contract.

4.5 Evaluation

In this section, I demonstrate the effectiveness of CoCo in uncovering the impact of DCWs. I obtained contracts marked as fraudulent from Etherscan [183]. To avoid false positives and potential overclaiming, I manually verified each marked contract by checking for either public reports or reported victims in Etherscan's comment section. This resulted in 157 flagged contracts. I utilized an open-source project [214] that pulled the Etherscan labeled data.

4.5.1 Post Deployment Dataset Highlights

Deploying CoCo on my dataset revealed an unnerving trend in fraudulent activities perpetrated through smart contracts. Given 157 flagged smart contracts on Etherscan [183],

CoCo uncovered a total of 1,283,198 associated contracts from 91 DCWs behind these contracts. The distribution of these associated contracts is presented in Figure 4.4. Furthermore, as highlighted under *Profit (ETH)* in Figure 4.4, CoCo assessed the impact of these fraudulent activities by calculating the ETH equivalent of the illicit profits stolen by DCWs. Specifically, CoCo determined that DCWs amassed a total of 2,638,752 ETH in illicit profits, averaging 2.06 ETH per contract. Interestingly, Figure 4.4 also reveals a strong correlation between the number of associated contracts and the profits accruing to DCWs. This suggests that rather than relying on a small number of contracts, DCWs are more inclined to distribute risk across various fraud contracts. The tactics of DCWs reflect the effectiveness of current mitigation strategies, such as flagging, in influencing DCWs' operations.

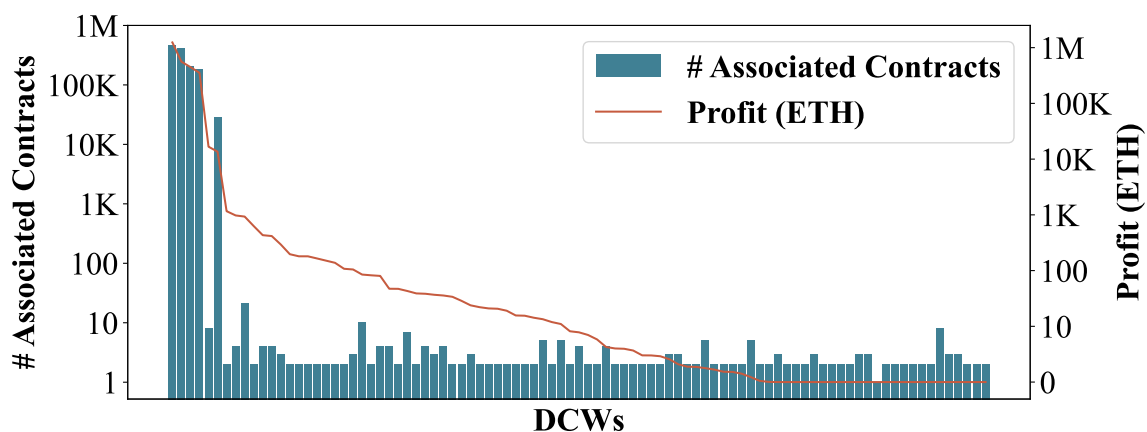


Figure 4.4: Distribution of associated contracts and illicit ETH profits across DCWs.

4.5.2 Impact Of DCW

Table 4.3 presents the top 10 DCWs with the highest fraud profit identified by CoCo. Column 1 shows the abbreviation of each DCW. Columns 2 and 3 show the start and end dates of each DCW's fraudulent activities, respectively. Column 4 details the number of fraudulent transactions associated with each DCW outlined in Column 1, arranged temporally. Column 5 indicates the aggregate quantity of ETH illicitly appropriated by each DCW. Considering

Table 4.3: Top 10 DCWS That Made Most Illicit Profit Through Fraud Contracts.

DCW	Start Time	End Time	Fraud Transaction Volume	ETH	USD	Victim			
						Min	Max	Avg	Total w CoCo
W-521058	Oct 2017	Jun 2023		1,240,355	1,384,080,119	1	2,978	471	232,011 209,926%
W-3cd202	Feb 2018	Jun 2023		280,053	192,665,735	1	2,842	303	173,469 113,519%
W-3a3250	Sep 2017	Jun 2023		447,891	141,686,276	2	5,756	295	74,074 116,699%
W-4b2c9b	Nov 2017	Jun 2023		13,605	9,639,322	1	99	16	5,833 5,382%
W-9989f8	Apr 2018	Jul 2018		16,706	7,976,355	1	5	0	43 1%
W-ec1ef1	Aug 2022	Feb 2023		976	1,374,083	1	714	67	14,489 0%
W-82f98b	Apr 2022	May 2022		416	1,224,971	1	14	5	93 1%
W-e058d6	Dec 2020	Feb 2021		931	1,138,740	1	11	2	130 17%
W-2499e7	May 2022	May 2022		432	901,275	1	8	4	51 2%
W-f09117	Sep 2017	Sep 2017		1,163	394,444	1	19	3	26 0%
Summary	Sep 2017	Jun 2023		2,002,533	1,741,081,326	1	5,756	232	493,068 445,556%

that the objective of the DCW is to generate real-world currency profits, CoCo converts the stolen ETH into USD based on the exchange rate at the time of each transaction; this converted amount is reflected in Column 6. Beyond assessing the economic impact to indicate the extent of the fraudulent activities, FBI agents should also consider the severity of the fraud based on the number of affected individuals. In support of this, CoCo identifies the associated contracts and their corresponding transactions, subsequently determining the *origin* of each transaction. Such origins are used for the estimation of victim accounts. Columns 7 through 9 show the minimum, maximum, and average number of *distinct* victims engaged with the DCW detailed in Column 1 on a daily basis. Column 10 provides a total victim count. To underscore the significance of CoCo, I compared the efficacy in victim identification by FBI agents solely from reported fraud contracts against the enhanced forensics capabilities equipped by using CoCo.

Table 4.3 provides insight into the operations of fraudulent activities conducted via smart contracts. Column 2 reveals that, out of 10 contracts analyzed by CoCo, the inception of such fraudulent activities dates back five years to September 2017, which is only two years subsequent to Ethereum's introduction [215]. This highlights the long-standing presence of fraudulent undertakings within the Ethereum network. It was anticipated that this prolonged history of fraudulent acts would have garnered the attention of law enforcement and government agencies, prompting them to initiate countermeasures. Unfortunately, when I examine Column 3 of Table 4.3, it is apparent that four out of 10 DCWs (40.00%) remain active up to the date of this study (shown as Jun 2023 in Column 3 of Table 4.3). For example, Row 1 of Table 4.3 illustrates that the fraud led by W-521058 has persisted for five years. Considering the observed efforts by government agencies (e.g., the FBI) to combat general cryptocurrency fraud [201], it is clear that the stealth and complexity of smart-contract-based fraud have outpaced the investigative capabilities of legal authorities.

The enduring presence of DCWs, coupled with inadequate investigative approaches, has led to the rampant presence of these fraudulent activities. An examination of Column 4

of Table 4.3, which details the daily transaction volume of the DCW listed in Column 1, yields several notable observations: (1) As evidenced by Rows 1-4 of Table 4.3, substantial DCWs do not opt for discretion; instead, they exhibit consistently high volumes of fraudulent transactions on a daily basis. (2) Although the transaction volume for less-aggressive DCWs may vary, as depicted by Rows 5-10 of Table 4.3, their brief duration of activity still demonstrates a significant level of traffic.

The high volume of fraudulent transactions directly correlates with the substantial profits these DCWs yield. Columns 5 and 6 of Table 4.3 demonstrate that the foremost 10 DCWs have collectively garnered 2,002,533 ETH (\$1,741,081,326) in illicit profits. In reviewing Row 1 of Table 4.3, it becomes evident that the most profitable scheme, conducted by W-521058, has amassed 1,240,355 ETH (\$1,384,080,119) in unlawful earnings, accounting for 61.94% of the total ETH and 79.50% of the total USD accumulated by these 10 leading DCWs.

It is clear that fraudulent activities facilitated by smart contracts present a critical issue, especially considering the substantial illicit profits they have accrued. Alarming, the situation may escalate when considering the number of victims impacted by these DCWs. Columns 9-10 of Table 4.3 detail the minimum, maximum, and average number of daily victims for the top 10 DCWs. A closer look at Column 10 of Table 4.3 shows that the total victim count from these fraudulent operations could reach 493,068. Focusing on Row 1 of Table 4.3, I find that the fraudulent activities conducted by W-521058 have affected 232,011 individuals, which represents 47.05% of all identified victims. Comparing Columns 5-6 with Column 10, it is apparent that, on average, each victim has lost 4.06 ETH (\$3531.12) to these DCWs. I have thus demonstrated that fraudulent activities executed via smart contracts are a great concern, both in terms of the illicit profits and the number of victims affected. To further underscore the need for more proactive forensic analysis, Column 11 of Table 4.3 shows the potential increase in identified victims using CoCo. The Summary row highlights an average gain of 445,556% additional victims, reinforcing the importance of integrating

Table 4.4: Top 15 Fraud Contracts Identified By CoCo That Make Highest Illicit Profits.

Contract	Start Time	End Time	Daily Revenue	ETH			USD		
				Min	Max	Avg	Min	Max	Total
C-cd154b	Oct 2017	Jun 2023		0.005	8,364	147	8.690	5,847,875	99,874 206,240,342
C-cc1d68	Feb 2018	Jun 2023		0.060	1,238	53	52.057	2,402,960	49,762 97,036,980
C-2f9ffe	Oct 2019	Jun 2023		0.004	666	42	7.383	1,029,244	60,650 81,635,779
C-e00b88	May 2021	Jun 2023		0.040	609	37	99.679	1,267,433	90,000 68,040,043
C-e2985f	May 2021	Nov 2022		0.010	14,344	47	29.309	23,519,179	84,345 45,546,477
C-f03cef	Mar 2022	Jun 2023		1.864	613	54	2236.746	955,632	82,739 39,053,042
C-6f5365	Dec 2019	Jun 2023		4.060	559	15	593.342	716,962	15,858 20,346,288
C-cedd45	Nov 2021	Jun 2023		0.095	397	32	410.665	1,161,698	64,659 38,795,418
C-5e0679	Dec 2017	Nov 2022		17.958	450	10	10,977.451	756,184	11,387 20,543,033
C-3c2551	Nov 2017	Nov 2022		0.630	192	9	640.500	606,352	22,067 40,074,910
C-af45d2	Nov 2022	Nov 2022		14,592	14,592	14,592	17,733,852	17,733,852	17,733,852
C-3c9b11	May 2021	Jun 2023		4.992	199	13	9,148.109	690,314	27,946 21,798,422
C-49c546	Apr 2021	Dec 2022		0.545	268	15	2,421.490	675,311	41,793 25,828,472
C-47961f	Oct 2021	Jun 2023		1.000	500	14	3,971.950	937,348	27,819 16,858,621
C-b65ed7	Nov 2017	Jun 2023		0.295	232	4	94.204	177,506	3,027 6,211,922
Summary	Oct 2017	Jun 2023		0.004	14,592	325	7.383	23,519,179	360,436 745,743,608

CoCo into DCW fraud forensics.

4.5.3 Drill Down Into Fraud Contracts

As detailed in §4.5.2, I have established that the illicit profits collected by the fraudulent activities can reach \$1,741,081,326 and the the number of involved victims can easily reach 493,068. However, it is unclear whether the smart contracts driving these frauds exhibit a similar transaction pattern (e.g., volume) or profit margin. To address this gap, I delve deeper into the contracts in the following section. Table 4.4 lists the top 15 fraudulent contracts, ranked by the highest illicit profits, as identified by CoCo. Column 1 shows the abbreviation for each contract; readers interested in the full address can refer to Table C.1. Columns 2-3 outline the start time and end time of the fraudulent activities for each contract, respectively. Column 4 presents the daily fraudulent transaction volume timeline for each contract in Column 1. Columns 5-7 show the daily illicit ETH profits for each contract in Column 1, formatted as Minimum (Min), Maximum (Max), and Average (Avg). Column 8 tallies the total amount of ETH each contract has accumulated from the transactions of victims. To further quantify the impact of these fraud contracts, CoCo has converted the ETH amounts into USD based on the historical exchange rates at the time of each transaction. The corresponding USD profits are shown in Columns 9-12. Specifically, Columns 9-11 detail the daily USD profits, while Column 12 aggregates the total USD revenue accrued by each contract throughout its period of activity.

Table 4.4 provides interesting insights into the fraud contracts that direct fraudulent activities orchestrated by DCWs. As discussed in §4.5.2, I observed that fraudulent activities tend to have an extended duration of activity. Meanwhile, an examination of Columns 2-3 in Table 4.4 indicates that 14 out of 15 contracts (93.33%) have sustained fraudulent activities for over a year. This observation could be attributed to the immutable characteristic of blockchain technology. Specifically, once fraud contracts are exposed, their perpetrators might not be concerned about the contracts being terminated. Intuitively, one

might assume that these fraud contracts would exhibit a consistently high volume of fraudulent activity since the orchestrators do not have to worry about the contracts being dismantled. Surprisingly, Column 3 of Table 4.4 shows that the contracts on Rows 4, 8-10, and 14-15 of Table 4.4 (40.00% of the top 15 contracts) undergo a significant period of dormancy before experiencing a resurgence of fraudulent activities. This pattern suggests that, regardless of their lack of concern over contract removal, orchestrators still take measures to prevent the fraud contracts from being detected.

Upon reviewing Columns 8 and 12 of Table 4.4, it is evident that the top 15 contracts have collectively accumulated illicit gains of 673,221 ETH (\$745,743,608). It is important to note that the USD profits are calculated using the historical ETH-to-USD exchange rate corresponding to the dates of the fraudulent transactions. Considering the extended duration of the fraudulent activities and the rising value of ETH, it is conceivable that the perpetrators might realize even greater profits if they were to liquidate their ETH holdings at current market prices. Delving deeper into the profit analysis, a focus on Columns 7 and 11 of Table 4.4, which outline the average daily ETH and USD profits generated by the contracts listed in Column 1, shows that periods of dormancy have not impeded the substantial accumulation of profits by the orchestrators. Referencing Table 4.4, it is evident that with the aid of CoCo, FBI agents can ascertain that the average daily profits of the contracts amount to 325 ETH and \$360,436. These figures constitute 2.23% and 1.53% of the maximum daily profits, respectively. The data highlighted here underscore the orchestrators' proficiency in sustaining a regular flow of illicit revenue, navigating even through sporadic phases of dormancy. This revelation amplifies the gravity of fraudulent activities facilitated by smart contracts and underscores the urgency to deploy CoCo for comprehensive forensic analysis.

4.6 Validation

I implemented a prototype of CoCo in Python leveraging Mythril [216], with customized module (~10K lines) to perform program analysis and on-chain transaction analysis (~10K

Table 4.5: CoCo’s VALIDATION.

Contract	DCW	Evidence ❶				Evidence ❷				Evidence ❸			
		Associated Contracts		Recipients		Deploy Length ²		Origin		GT		FP	FN
		GT	C ¹	GT	C	GT	C	GT	C	GT	C		
C-98f804	W-521058	461,015	461,015	1	1	987,360	987,360	1	1	2	2	0	0
C-6113fB	W-bceE76	1	1	1	1	2	2	1	1	1	1	0	0
C-Fd562c	W-82f98B	3	3	1	1	6	6	1	1	1	1	0	0
C-2e14CC	W-2499E7	3	3	1	1	6	6	1	1	1	1	0	0
C-8f428e	W-b10f4B	2	2	1	1	4	4	1	1	1	1	0	0
C-18B228	W-3CD202	406,559	406,560	1	1	1,626,234	1,626,236	1	0	1	2	4	1
C-805a29	W-6629BA	2	2	1	1	4	4	1	1	1	1	0	0
C-078228	W-2b0878	2	2	1	1	4	4	1	1	1	1	0	0
C-0E712A	W-09739F	1	1	3	3	2	2	3	3	1	1	0	0
C-db7aFF	W-187307	3	3	3	3	7	7	3	3	3	3	0	0
C-70aae3	W-1AB876	4	4	2	2	10	10	2	2	4	4	0	0
C-e18895	W-5FFDdc	183,108	183,109	1	1	549,324	549,326	1	1	1	1	3	0
Total	12	1,050,703	1,050,705	17	17	3,162,963	3,162,967	17	16	18	19	7	1

1: C is short for CoCo.

2: Deploy Length refers to the number of accounts involved in the deployment of a contract.

This column shows sum of Deploy Length of all associated contracts.

lines). I conducted my experiments on an Ubuntu 20.04 LTS system equipped with 12 CPUs and 64GB of memory. For validation purposes, I randomly selected fraud contracts from my dataset until I found 12 with different DCWs. The distribution of ground truth dataset aligns with the overall distribution of the DCW and the number of fraud contracts they deployed, shown in Figure 4.4 in §4.5.1. Notably, since contract flagging serves as a user warning and removing these accounts from the blockchain is almost impossible, these flagged contracts might still be active. To ensure the reproducibility of my study, I based my analysis on the Ethereum state as of June 26, 2023.

Ground Truth. Generating ground truth for the validation dataset required me to produce both on-chain transaction ground truth and off-chain contract implementation ground truth. For transaction ground truth, I deployed an Ethereum archive node utilizing Reth [217] to access the on-chain data. For the contract implementation ground truth, I employed Panoramix [218] and Dedaub [219] to decompile each contract, followed by a manual inspection of the decompiled code to ascertain its provenance, logic, and capabilities.

4.6.1 Associated Contracts & Recipients

Table 4.5 presents CoCo’s validation result. As shown in the False Positive (FP) and False Negative (FN) columns, CoCo generated seven FPs and one FN. To prevent overstating performance, I assign one TP in both Associated Contracts (Columns 3-4 of Table 4.5) and Deploy Length (Columns 7-8 of Table 4.5) if CoCo correctly identifies all associated contracts and deployment lengths, respectively, otherwise assigning zero in each case. Given this, it can be seen that CoCo achieves an overall accuracy of 90.91%, demonstrating its efficiency in generating precise forensic evidence. Delving further into the performance of CoCo, each piece of evidence (Evidence ❶, Evidence ❷, and Evidence ❸) is evaluated. The first two columns of Table 4.5 list the abbreviation of each reported fraud contract processed by CoCo and the uncovered DCW, respectively. Interested readers can find the

full address of the abbreviation in Table C.1. Columns 3-6 show the validation results for Evidence ①.

As shown in the Total row, CoCo pinpointed 1,050,705 associated contracts, along with 17 recipients. My ground truth data indicates the presence of 1,050,703 associated contracts with 17 recipients. Upon a thorough investigation, I found that two FPs can be attributed to the distinct methods employed by DCWs (Row 6 and Row 12 of Table 4.5) in deploying the fraud contracts. Specifically, CoCo discovered that W-3CD202 and W-5FFDdc invoked contract C-36FFaE and C-1e3e4e to introduce new fraud contracts, respectively. In the course of analyzing C-36FFaE, CoCo identified that, rather than employing the standard CREATE or CREATE2 operations for contract deployment, C-36FFaE instead delegated this task to an alternative contract, C-2C4691. Consequently, CoCo flagged C-2C4691 as associated contracts. However, a manual review of C-1e3e4e and C-2C4691 revealed that they were in fact deployed by different accounts, W-53C9dA and W-13666c, respectively. Considering that FBI agents in this case would have no obvious evidence of the fraud intention, I consider this as FP. Notably, CoCo still reported these two contracts since they give FBI agents an important lead on potential contract abuse. I confirmed that these are rare cases in my dataset. Overall, CoCo was 90.91% accurate in generating evidence, making it robust for my evaluation.

4.6.2 Dynamically Resolved Recipients

Columns 7-10 of Table 4.5 present the validation result of CoCo generating Evidence ②. As illustrated in Column 8 and Column 10 of Table 4.5, CoCo identified 3,162,963 Deploy Lengths (number of accounts involved in the deployment of a contract) and 17 origins of recipients. As shown on Row 6 and Row 12 of Table 4.5, CoCo generated FP when analyzing DCW W-3CD202 and W-5FFDdc. This FP is the direct result of the FP in the Evidence ① generation, as discussed in §4.6.1. I also see one False Negative (FN) shown on Row 6 of Table 4.5 (W-3CD202) during the track of the recipient's origin. As discussed in §4.6.1,

when W-3CD202 invokes the intermediate contract C-36FFaE, it delegates the responsibility of deploying fraud contracts to C-2C4691. By manually dissecting the implementation of C-36FFaE, I found that the function it will trigger in C-2C4691 is not hard-coded. Instead, it is resolved based on the transaction invoking C-36FFaE. Consequently, the absence of a static entry point hindered CoCo's capacity to pinpoint the contract deployment process used by the DCW, W-3CD202, thereby impeding the tracing of recipient origins in contracts deployed by C-2C4691. Nevertheless, as discussed earlier, such instances are rare cases. A 90.91% accuracy in generating evidence makes CoCo robust for my evaluation.

4.6.3 Capability Attribution

Columns 11-12 illustrate the performance of CoCo regarding the generation of Evidence ③. As presented in the Total row, CoCo has successfully grouped 18 associated contracts into attributed capabilities from the 12 fraud contracts. Nonetheless, my verified ground truth indicates 19 such groups. Further scrutiny, as discussed in relation to the earlier FP incident detailed in §4.6.1, reveals that during the analysis of the transactions originating from the fraud contract C-36FFaE (shown on Row 6 of Table 4.5), CoCo included contract C-2C4691 because it was designated by C-36FFaE to carry out the deployment of fraud contracts. Since C-2C4691 operates as a multi-signature wallet contract, CoCo misclassified it as a separate attributed group. It's important to note that the FP in the Evidence ③ generation is intrinsically linked to the FP in the Evidence ① generation. FBI agents, by identifying and excluding C-2C4691 from the forensic analysis of the DCW, can eliminate this error in the generation of Evidence ③ as well. Considering the minimal occurrence of FPs and FNs, coupled with an accuracy rate of 90.91% , CoCo is validated as an effective tool for generating the three pieces of evidence of the DCW that FBI agents require to mitigate fraudulent activities.

4.7 Discussion

Ethical Concerns. The data utilized in this analysis is sourced exclusively from the public domain, specifically Ethereum. This research centers on conducting forensic investigation of fraud contracts, and it is important to clarify that I *do not* engage in any exploitative activities.

Challenges of Symbolic Analysis. CoCo leverages symbolic analysis, a technique adopted by top-tier research [1], [32], [71], [84], [85], [86]. While symbolic analysis often faces the challenge of path explosion, my evaluation in §4.5 shows this to be a rare issue in smart contract analysis. This is likely because smart contract execution involves a cost, known as ‘gas,’ leading malicious actors to prefer simpler contract designs.

ERC-20 & NFT. CoCo primarily focuses on the forensic analysis of fraudulent activities involving the scam of ETH. ERC-20 [220] and Non-Fungible Tokens (NFTs) [221] have emerged as extensions built upon Ethereum. ERC-20 tokens provide a standardized protocol for fungible tokens, ensuring consistency in token interactions. Non-Fungible Tokens (NFTs), in contrast, represent unique digital assets, each characterized by distinct metadata and individual ownership records. Even though both token standards are implemented through smart contracts, forensic analysis of ERC-20 and NFT requires distinct approaches. In particular, both transaction and program analysis offer limited insight into the complex behaviors associated with these tokens. While ETH transfers are directly recorded as transactions, exchanges of ERC-20 tokens or activities like NFT minting and ownership transfer are typically reflected as internal state changes within the contracts themselves.

Generalizability of CoCo. The design principles of CoCo are not tied to specific features of any blockchain (e.g., Ethereum). This makes CoCo portable to other blockchain platforms. Specifically, since blockchain technology mandates the public disclosure of all transactions, the transaction forensics components of CoCo can be applied across different

blockchains. Additionally, the program analysis components of CoCo can be easily extended for use with any blockchain that supports smart contracts. For instance, BNB Smart Chain (BSC) [222] is compatible with EVM bytecode, enabling direct application of CoCo. Similarly, Solana [223] employs Berkeley Packet Filter (BPF) bytecode for its smart contracts. Porting CoCo for Solana only requires a one-time effort to map the current bytecode set used by CoCo to Solana’s bytecode set.

4.8 Related Work

4.8.1 Fraud Detection On Blockchain

Ponzi Scheme. Yu et al. [184] utilized graph convolutional networks to detect Ponzi schemes using transactions, whereas Zhang et al. [185] improved on the LightGBM algorithm for better detection efficacy. Extracting bytecode features [186] and leveraging text convolutional neural networks [187] have also been proposed to identify Ponzi-like characteristics in smart contracts. There are also works using opcode compression for vulnerability detection [188] and identifying static features within smart contracts for Ponzi scheme classification [189]. Techniques using opcode sequences [190], code analysis [191], and enhanced convolutional neural networks [192], [193] have also been implemented to identify Ponzi schemes.

Fraud And Phishing Detection. Beyond Ponzi schemes, broader fraud detection on the blockchain is tackled through various innovative approaches. Liu et al. [195] utilized a Heterogeneous Information Network to model smart contracts and apply graph Transformer networks to detect abnormalities. Machine learning algorithms, such as XGBoost and Random Forest, have been employed by Ashfaq et al. [196] to classify Ethereum transactions and detect anomalies like double-spending and Sybil attacks. Furthermore, Hu et al. [197] explored deep learning models to identify scams on a large scale by examining the N-gram bytecode patterns of a smart contract. Notably, the identification of fraud contracts acts as

the input for CoCo, enabling it to generate evidence that assists FBI agents in obtaining legal authorization and implementing mitigation strategies. The research community has also studied a wide range of Phishing transactions [224], [225], [226], [227]. Unlike these existing studies, which treat the fraud contracts like a blackbox and focus on examining/triggering transactions, CoCo bridges the gap between the contract implementation and on-chain transactions to produce forensic evidence.

4.8.2 Smart Contract Vulnerability

Luu et al. [202] were among the first to address the security risks in smart contracts by identifying security pitfalls. Symbolic analysis has been used frequently to detect such vulnerabilities; Krupp and Rossow [203] developed techniques to automatically exploit smart contracts by analyzing bytecode. Similarly, Zeus [204], a framework for the symbolic verification of smart contracts, uses Constrained Horn Clauses to identify vulnerabilities, an approach also adopted by others [205], [207]. With the rising complexity of smart contracts, especially those involving multiple contracts, Ma et al. [206] proposed an inter-contract analysis tool, while others have focused on state inconsistency bugs [208]. Online detection methods have also been investigated, such as SODA [228], an online detection framework for smart contracts.

Recent works have shifted toward leveraging machine learning to enhance the detection process. Sendner et al. [229] introduced ESCORT, a method employing deep learning to identify different types of smart contract vulnerabilities. There are also works utilizing dynamic analysis [230], deep learning [231], and pre-training techniques [232] to identify smart contract vulnerabilities. Others have focused on static analysis rules tailored to specific blockchain platforms, like VRust for Solana smart contracts [233], or on defining critical paths to address fund transfer vulnerabilities [234]. Moreover, combining expert knowledge with graph neural networks has shown promise in enhancing detection capabilities [235]. Another notable approach is the analysis of confused deputy

vulnerabilities in Ethereum smart contracts [236], which highlights the importance of understanding the security challenges in contract interactions. Contributing to a different domain, CoCo conducts forensic analysis on fraud contracts, producing the evidence for FBI agents to obtain legal authorization and execute mitigation strategies.

4.9 Conclusion

Applying CoCo to 157 Etherscan-flagged contracts [183], my research identified 1,283,198 associated contracts across 91 DCWs. These frauds have accrued 2,638,752 ETH (\$2,089,504,682) in illicit profits, averaging 2.06 ETH (\$1628.36) per contract. CoCo revealed that these frauds date back to September 2017. Notably, my research found that scammers tend to employ multiple fraud contracts to distribute the risk, suggesting the efficacy of current flagging mitigation strategies upon scammers. In response, I am actively collaborating with Etherscan [183] and the FBI [209] to take actions based on my findings.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

5.2 Future Work

My future research agenda follows a bottom-up approach. I plan to advance core program analysis techniques, create new methods that automate proactive forensics as cyberattacks evolve, and establish a scientific basis for converting extracted evidence into real-world remediation. Achieving this vision will require multi-disciplinary collaboration with industry partners and colleagues who bring expertise in security, economics, and law enforcement.

5.2.1 Detection of Malicious Traffic Targeting Web Applications

Prior work in this thesis—MARSEA and VADER—uses domain-specific knowledge such as behavioral models of web applications to interpret malicious activity. Yet the growing diversity of online platforms makes this strategy hard to scale for timely remediation. Future systems must detect and remediate web application abuse while acquiring and adapting domain knowledge as attackers shift to new environments.

Interestingly, when adversaries exploit benign services, they must operate within public policies and technical documentation. These materials are written in natural language and provide a valuable source of structure for automation. Future work will use Large Language Models (LLMs) to parse, formalize, and encode this documentation into machine-readable specifications that support context-aware program forensics. This line of work has already begun in collaboration with industry partner Netskope, where HTTP endpoint documentation is used to model the semantics of malware-generated traffic. Advancing this agenda will require collaboration with colleagues who have expertise in LLM reasoning.

5.2.2 Emerging Threats in AI-Enabled Cybercrime

As AI systems become part of daily workflows, adversaries are beginning to weaponize them. They now automate reconnaissance, generate polymorphic payloads, and craft scam messages. Future research will extend proactive forensics into this emerging AI-enabled threat landscape through empirical studies and program analysis to characterize how cybercriminals abuse AI services. This work will produce public datasets and taxonomies of AI-driven campaigns.

Future work will also develop cross-platform analysis techniques that combine program analysis, network forensics, and Open Source Intelligence (OSINT) to attribute AI interactions to the online personas behind these attacks. To support more proactive investigations, probabilistic modeling will be used to link these personas to real-world entities. The goal is to develop attribution techniques that provide law enforcement with reliable, prosecutable links between digital activity and the individuals behind cybercrime. This research requires continued collaboration with law enforcement partners such as the U.S. FBI and multidisciplinary collaborators working on LLM safety, adversarial machine learning, and automated reasoning for security.

5.2.3 A Unified Science of Proactive Cybercrime Remediation

Despite progress in detecting and remediating individual incidents, cyber forensics still evaluates success at the technical level—isolated takedowns or blocked connections—and rarely measures impact at the ecosystem level, such as whether campaigns are deterred or simply reorganized. The long-term goal is to build a science of proactive cybercrime remediation that can digest emerging threats, extract actionable evidence, generate remediation strategies, and synthesize effective detection methods. This science must also connect technical forensics with economic incentives and legal outcomes.

Building on systems that already produce evidence for law enforcement and support real-world takedowns, future work will create models that quantify how remediation actions

change adversaries' costs and risks, helping defenders choose actions that maximize durable deterrence rather than short-lived suppression. This research agenda aligns with interdisciplinary funding initiatives such as NSF SaTC, the NSF Social, Behavioral and Economic Sciences (SBE) programs, and mission-driven programs at DARPA and the Department of Justice, and will be pursued through collaborations with colleagues who study the human dimensions of security, political science, and cybersecurity economics.

Appendices

APPENDIX A

MARSEA SUPPLEMENTARY MATERIAL

A.1 Web Apps And Available Information

Table A.1: Web Apps And Available Information

Category	Web Apps	Identity	Assets	CT ¹	UT ¹	OA ²	Views
File Sharing	GDrive	Email	File	✓	✓	-	-
	Dropbox	Username	File	✓	✓	-	-
	Pastebin	Username	Post	✓	✓	✓	✓
	Zippyshare	-	File	✓	-	-	-
	Imgur	Username	Image	✓	-	✓	✓
	Tietuku	Username	File	✓	-	-	-
Software Hosting	GitHub	Username	Repo/File	✓	✓	✓	-
Social Engineering	Telegram	Username	Channel	✓	✓	-	✓
	Twitter	Username	Tweet	✓	✓	✓	-
	Discord	Channel	File/Hook	✓	✓	-	-
	Slack	Channel	Hook	-	-	-	-
	Facebook	Username	File/Text	✓	✓	✓	-
	V kontakte	Username	File/Text	✓	✓	✓	✓
Website Builder	WordPress	Username	Web Page	✓	✓	✓	-
	Webs	Username	Web Page	✓	✓	✓	-
	Tripod	Username	Web Page	✓	✓	✓	-
Blog	Blogspot	Username	Web Page	✓	✓	✓	-
Dynamic DNS	Afraid	API key	DNS record	✓	✓	-	-
	DuckDNS	API key	DNS record	✓	✓	-	-
Anti-Virus	VirusTotal	API key	-	-	-	-	-

1: Creation Time (CT) and Update Time (UT).

2: Other assets from the same identity.

Table A.1 presents the web apps Marsea found during my study and the information publicly available from each web app. Columns 1-2 present the web app category and the corresponding web apps. Columns 3-4 present the identity and assets Marsea supported to identify. Columns 5-8 show whether the web apps expose asset Creation Time (CT),

Update Time (UT), other assets from the same identity (OA), and the views of the assets, respectively.

Please note that as stated in §2.5.2, Local Info web apps shown in Column 1 of Table 2.4 have no concept of identity and assets, thus, are not included in Table A.1. Interestingly, as shown in Row Rows 10-11 of Table A.1, Marsea found that WAE malware abuse webhook. However, Slack does not expose information to the public, given only a webhook. Unlike Slack, as shown in Row 11 of Table A.1, Discord revealed the channel ID in the webhook endpoint, through which Marsea could harvest *CT* and *UT*. As shown in Columns 7-8, only 10 (50%) and 4 (20%) web apps expose the OA and views of assets to the public. Given the importance of OA and views of assets when IRs and web app providers pursue the *Lateral Remediation* (§2.5.3) and extent of infection evaluation, making these data publicly available or at the early stage of collaboration should be encouraged. Meanwhile, given Marsea's ability to identify vectors and attribute sessions to them, IRs could still collaborate with the web app providers.

A.2 Response From Web App Providers

Figure A.1 presents my communication with Discord when they experienced high reporting volume, and Figure A.2 shows the takedown confirmation by Discord. Figure A.4 shows the takedown confirmation by MediaFire. Figure A.3 shows the case where GitHub needed 103 days to initialize the investigation.

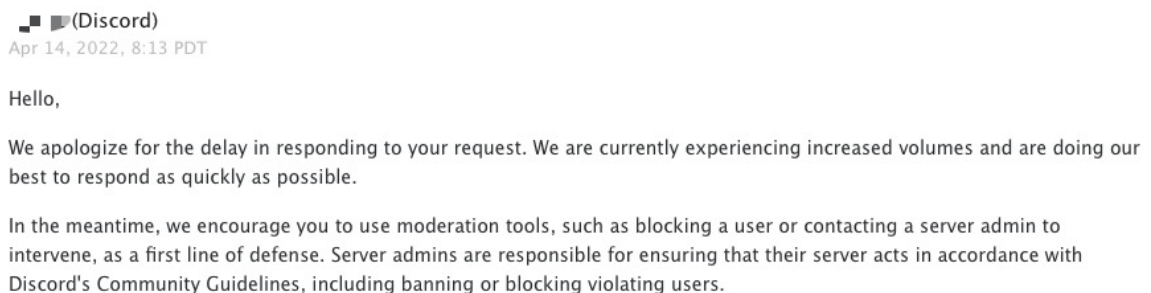


Figure A.1: Late Response From Discord.

 (Discord)

Apr 6, 2022, 11:19 PDT

Hello,

Thank you for bringing this issue to our attention. We've initiated an investigation based on the information that you provided and we'll take appropriate action based on our findings. Please note that for privacy reasons, we're not able to share the specifics of the action taken, if any.

We truly appreciate your efforts in helping us to keep Discord a safe and friendly environment.

Sincerely,
Discord Trust & Safety

Figure A.2: Confirmation By Discord.



GitHub Trust & Safety (GitHub Support)

Jul 18, 2022, 1:17 PM UTC

Hi,

Thanks for taking the time to let us know. Our team is currently investigating the account in question to determine if their content or conduct violates GitHub's [Terms of Service](#).

Please let us know if we can help in any other way.

Regards,
GitHub Trust & Safety





Apr 6, 2022, 3:13 PM UTC

Figure A.3: Response From GitHub.



██████████ (MediaFire)

Apr 5, 2022, 3:16 PM CDT

Hi

Thank you for contacting MediaFire. This has been processed.

Best Regards,



MediaFire | Customer Support

Figure A.4: Response From MediaFire.

APPENDIX B

VADER SUPPLEMENTARY MATERIAL

B.1 Anti-Analysis APIs

Malware uses anti-analysis and defensive evasion APIs to avoid detection. VADER hooks these APIs to facilitate malware exploration. The APIs used in VADER’s design are listed in VADER’s code repository [237]. These APIs were identified through manual malware analysis, industry reports [36], [145], and prior research on system emulation and evasion techniques [146], [147], [148].

B.2 De-manipulation Algorithm Source Code Similarity

I use Moss [143] to confirm that each decoding algorithm in VADER is distinct. The comparison results are available in VADER’s code repository [238]. As shown in Row 6, no algorithms overlap, where $|R|$ is the set of all de-manipulation algorithms, and m, n index different algorithms. Matches occurred only when an algorithm was compared with itself. This confirms that each de-manipulation algorithm in VADER is uniquely implemented.

B.3 De-Manipulation Comparison Graphs

I made the symbolic expression of de-manipulation algorithm comparisons available in VADER’s code repository [239]. For example, Column 1, Row 1 shows comparisons for three Base16 versions, totaling nine comparisons (e.g., Base16v1 vs. Base16v2, etc.). Each graph covers a two-hour runtime (X-axis), with lines representing de-manipulation similarity (Y-axis). A black vertical line marks the latest plateau, with all comparisons plateauing for over 30 minutes. Plateaus occurred between 20 and 85 minutes. I selected the best algorithm from each class for VADER to identify malware de-manipulation algorithms.

B.4 Web App Accounts Identified In My Study

The dead drops identified by VADER in this research have been made publicly available in VADER's code repository [240].

APPENDIX C

COCO SUPPLEMENTARY MATERIAL

C.1 Start Of Fraudulent Campaign

As highlighted in §4.2.2, FBI agents were notified with a fraud contract C-98f804 associated with a scam message on Telegram [14]. Further investigation into this message led agents to uncover similar content on other platforms, as shown in Figure C.1 and Figure C.2, reinforcing the message's fraudulent nature.

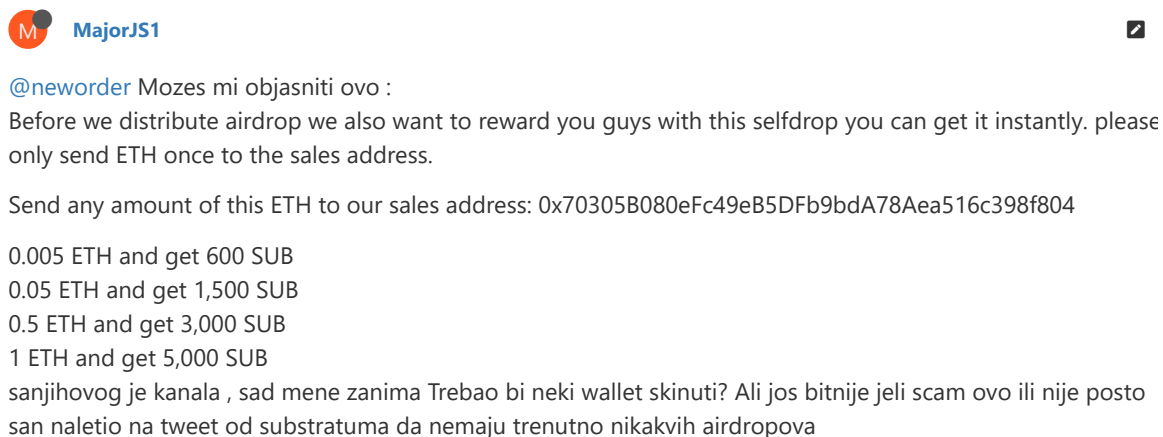


Figure C.1: Scammers post the fraudulent message on forum.

C.2 Collaboration With Investigators

Figure C.3 presents my communication with the FBI [209]. I redacted the name of the agent but kept the domain name for security and ethical considerations.

C.3 Full Forms Of Abbreviations

Accounts on Ethereum (i.e., wallets and contracts) are identified by 40-character hexadecimal strings known as addresses, while transactions are distinguished by 64-character hexadecimal

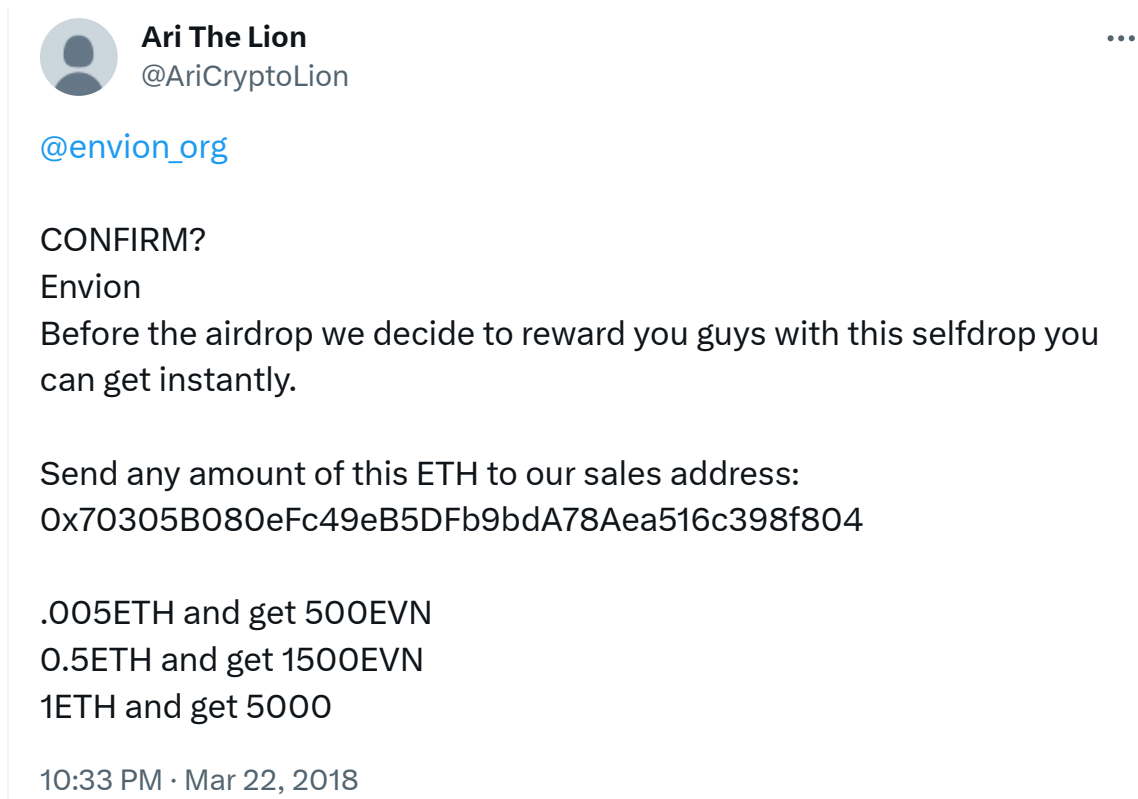


Figure C.2: Scammers post the fraudulent message on X.com.

strings, called transaction hashes. To improve the readability, I use abbreviations comprising the last six characters of addresses and transaction hashes in the paper. The full forms of these abbreviations are presented in Table C.1.

From: [REDACTED] <[REDACTED]@FBI.GOV>
Sent: Tuesday, December 5, 2023 10:50 AM
To: Saltaformaggio, Brendan D <brendan@ece.gatech.edu>
Subject: FBI Referral

Good Morning Brendan,

I received your contact information from a complaint you submitted involving fraudulent Ethereum smart contracts. I would love to set up a meeting to discuss the information you and your team identified during your investigation. Feel free to give me a call to discuss in detail and we can schedule a day in the next week to come to your office.

Thanks

Special Agent [REDACTED]
Complex Financial Crimes
FBI Atlanta
3000 Flowers Road South
Atlanta
Cell #: [REDACTED]
Desk #: [REDACTED]

From: Saltaformaggio, Brendan D <brendan@ece.gatech.edu>
Sent: Tuesday, December 5, 2023 1:18 PM
To: [REDACTED] (AT) (FBI) <[REDACTED]@FBI.GOV>
Cc: Yao, Mingxuan <mingxuanyao@gatech.edu>
Subject: [EXTERNAL EMAIL] - Re: FBI Referral

Hi [REDACTED],

Nice to talk to you. I am also looping in Mingxuan Yao (cc'ed). He is the lead researcher on this project.

We are free next Tuesday or Wednesday. Let me know if that works for your team.

We are in the coda building (<https://maps.app.goo.gl/LMwZGLqgrSekkBZ9>) in Midtown Atlanta.

Feel free to call my cell any time: [REDACTED]

Best,
Brendan

RE: FBI Referral

[REDACTED]@FBI.GOV>

Wed 12/6/2023 2:25 PM

To: Saltaformaggio, Brendan D <brendan@ece.gatech.edu>
Cc: Yao, Mingxuan <mingxuanyao@gatech.edu>

Good Afternoon Brendan,

Wednesday afternoon next week works for us. Let me know if that works for you all.

Thanks

Special Agent [REDACTED]
Complex Financial Crimes
FBI Atlanta
3000 Flowers Road South
Atlanta
Cell #: [REDACTED]
Desk #: [REDACTED]

Figure C.3: Collaborating with FBI.

Table C.1: Map Abbreviation To Full Forms.

Abbr	Full Forms
Contract	
C-98f804	0x70305b080efc49eb5dfb9bda78aea516c398f804
C-48d304	0x5BE1De8021cc883456FD11DC5CD3806dBc48D304
C-6113fB	0xf97Bd29b8eE6E246Eb57eEcf5D0E8486366113fB
C-Fd562c	0xefef14C36C1F2de2ca3772Ba9539B6A58cFd562c
C-2e14CC	0xcB3315A42E76b70D2f3e8E595a5d13855c2e14CC
C-789332	0xcf50193c27DF08423BFe813676541B2268789332
C-D986ae	0x8014FB4882b1f99a3E60AEcd1d39400560D986ae
C-6786ae	0x8014ae6574CAcE1f2435a86d4ea0472f466786ae
C-2D3B2f	0x65a8135596AE13C0Dd5c17bA1059C61Bc42D3B2f
C-D3c82b	0xDD499857c8539bEF04477B52782bE6A9FbD3c82b
C-159624	0xCC326C1D41f64c5331bc7Ba555d75306C3159624
C-8f428e	0xA77db707916aDEff81042ca57656931CcD8f428e
C-805a29	0x5F856630adBC27c0F5bC1DE1961D4f0fB1805a29
C-078228	0x1bd913BBaDE46bF5AD8b1e5d117701fBEb078228
C-0E712A	0xc25ab34E7F3a1eb2C6a3a23DF851F351df0E712A
C-a5f260	0xEb411D5Df13AC7020992306e78955fb7CBa5f260
C-db7aFF	0xBCC6C0feF89b87a12773Db7a9a8ECBCCCcdB7aFF
C-18B228	0x95115419B09E8Cea70a9bdbCA3fEe8C5e118B228
C-57e2e8	0x890bcE348BAE449Df3783ba0E1C7eB82C557e2e8
C-4f99A8	0x6032D639E634E788FcE323B316E06d18194f99A8
C-a5CDE9	0x6574C0bf7F3D144F5837acC160773eC8f2a5CDE9
C-3D7D37	0x197e45d545F4DA0C3f15002222BcADDd9D3D7D37
C-70aae3	0x4a96e9b57a229d94c0c28950355A72Fa9e70aae3
C-CbEB9E	0xfdd46E0ea17622d70AdaE6535948776160CbEB9E
C-e18895	0x3cD6ef508c1c448e293075f1dE2ae96a49e18895
C-2C4691	0x5B9E8728E316bBEB692d22daaAB74F6cBF2C4691
C-36FFaE	0x131A99859a8bfa3251D899F0675607766736FFaE
C-1e3e4e	0xbf0c5d82748ed81b5794e59055725579911e3e4e
Wallet	
W-521058	0x2E05A304d3040f1399c8C20D2a9F659AE7521058
W-bceE76	0x29203118cCbBF5277C1CEB49aF1333A91CbceE76
W-82f98B	0xd6E56a65f795Fd136406e668c0eB69360F82f98B
W-2499E7	0xA40b913D654D803b9833e9a699D5830f262499E7
W-b10f4B	0xF52426340e0548a8d58b970f2283e22c1bb10f4B
W-6629BA	0xD0E680D5141f6E61E953903736E3637a6E6629BA
W-2b0878	0xF7cC855E3BB2986729eC47c1B6f64e36aA2b0878
W-09739F	0x848a757656650c9950fb1Aed03eaC8A92209739F
W-223DcF	0xc2221f38dE2eB19125A5b77b5D82d5bFc7223DcF
W-187307	0x4005de995109895BE7Eac74346a62Db28b187307
W-3CD202	0x38c7eA86c8235b0CfCcFb91153259e85353CD202
W-297f83	0x7d3Bdf1b728386efDb9a3A0328a95D94b0297f83
W-15467A	0x7734aA368Df7bd09D2AbCBf925Cc92314A15467A
W-64EE46	0xfa32e18Cf5e9E96eDBa979f40DC55E465864EE46
W-1AB876	0xe3172Dc735B44893303e2fd22D1d3647271AB876
W-93abb5	0x9D31e30003f253563Ff108BC60B16Fdf2c93abb5
W-5FFDdc	0xd5e015739a8BEffF075C4eAA2013D27Df35FFDdc
W-13666c	0x058251232C086247cA91998472245D8Ae213666c
W-53C9dA	0x604Df452158e7ddF3E44338308EdC079a953C9dA
Transaction	
T-8bbae5	0x58f6fb1d2440eb4d6d5ac64a152aa156d3850eff6b3-56ab86904ac28758bbae5

REFERENCES

- [1] M. Yao, J. Fuller, R. P. Kasturi, S. Agarwal, A. K. Sikder, and B. Saltaformaggio, “Hiding in plain sight: An empirical study of web application abuse in malware,” in *Proceedings of the 32nd USENIX Security Symposium (Security)*, Anaheim, CA, Aug. 2023.
- [2] J. Fuller*, M. Yao*, S. Agarwal, S. Barua, T. Hirani, A. K. Sikder, and B. Saltaformaggio, “Enhanced Web Application Security Through Proactive Dead Drop Resolver Remediation,” in *Proceedings of the 32nd ACM Conference on Computer and Communications Security (CCS)*, Taipei, Taiwan, Oct. 2025, Acceptance rate: 13.8%. ***Co-first authors.**
- [3] M. Yao, R. Zhang, H. Xu, R. Chou, V. C. Paturi, A. K. Sikder, and B. Saltaformaggio, “Pulling Off The Mask: Forensic Analysis of the Deceptive Creator Wallets Behind Smart Contract Fraud,” in *2024 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, 2024, pp. 1651–1667.
- [4] *Cloud CDN: Content Delivery Network*, <https://cloud.google.com/cdn/>, [Accessed: 2022-04-05].
- [5] *Azure Content Delivery Network*, <https://azure.microsoft.com/en-us/services/cdn/>, [Accessed: 2022-04-05].
- [6] *Content Delivery Network (CDN) - Amazon CloudFront*, <https://aws.amazon.com/cloudfront/>, [Accessed: 2022-04-05].
- [7] *GitHub*, <https://github.com/>, [Accessed: 2022-04-15].
- [8] Fortinet, *The Malicious Use of Pastebin*, 2019. [Online]. Available: <https://www.fortinet.com/blog/threat-research/malicious-use-of-pastebin>.
- [9] *Dropbox*, <https://www.dropbox.com/>, [Accessed: 2021-04-06].

- [10] *Google Drive*, <https://www.drive.google.com/>, [Accessed: 2021-04-06].
- [11] *Amazon S3*, <https://aws.amazon.com/s3/>, [Accessed: 2021-04-06].
- [12] *Twitter*, <https://twitter.com/>, [Accessed: 2021-04-06].
- [13] *Instagram*, <https://www.instagram.com/>, [Accessed: 2021-04-06].
- [14] *Telegram Messenger*, <https://telegram.org/>, [Accessed: 2021-04-06].
- [15] *Turla Hacking Group Launches New Backdoor in Attacks Against US, Afghanistan*, <https://www.zdnet.com/article/turla-hacking-group-launches-new-backdoor-in-attacks-against-us-afghanistan/>, [Accessed: 2022-04-05].
- [16] *Russian Hacking Group Uses Dropbox to Store Malware-stolen Data*, <https://www.bleepingcomputer.com/news/security/russian-hacking-group-uses-dropbox-to-store-malware-stolen-data>, [Accessed: 2022-04-05].
- [17] *Turla's Crutch Backdoor Leverages Dropbox in Espionage Attacks*, <https://threatpost.com/turla-backdoor-dropbox-espionage-attacks/161777/>, [Accessed: 2022-04-05].
- [18] *Iranian Hackers Abuse Slack For Cyber Spying*, <https://www.forbes.com/sites/thomasbrewster/2021/12/15/iran-backed-hackers-use-slack-for-cyber-espionage>, [Accessed: 2022-04-05].
- [19] H. Badis, G. Doyen, and R. Khatoun, "Understanding botclouds from a system perspective: A principal component analysis," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Krakow, Poland: IEEE, May 2014.
- [20] H. Wang, Z. Xi, F. Li, and S. Chen, "Abusing public third-party services for edos attacks," in *Proceedings of the 10th USENIX Workshop on Offensive Technologies (WOOT)*, Austin, TX, Aug. 2016.

- [21] N. Pantic and M. I. Husain, “Covert Botnet Command and Control Using Twitter,” in *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC)*, Los Angeles, CA, Dec. 2015.
- [22] G. Lingam, R. R. Rout, D. V. L. N. Somayajulu, and S. K. Das, “Social Botnet Community Detection: A Novel Approach based on Behavioral Similarity in Twitter Network using Deep Learning,” in *Proceedings of the 15th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Taipei, Taiwan, Oct. 2020.
- [23] X. Liao, S. Alrwais, K. Yuan, L. Xing, X. Wang, S. Hao, and R. Beyah, “Cloud repository as a malicious service: Challenge, identification and implication,” vol. 1, no. 1, p. 14,
- [24] S. Gheewala and R. Patel, “Machine learning based twitter spam account detection: A review,” in *2018 Second International Conference on Computing Methodologies and Communication (ICCMC)*, IEEE, 2018, pp. 79–84.
- [25] S. Lee and J. Kim, “Warningbird: A near real-time detection system for suspicious urls in twitter stream,” vol. 10, no. 3, pp. 183–195,
- [26] A. Sanzgiri, A. Hughes, and S. Upadhyaya, “Analysis of malware propagation in twitter,” in *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*, 2013, pp. 195–204.
- [27] F. Concone, A. De Paola, G. L. Re, and M. Morana, “Twitter analysis for real-time malware discovery,” in *2017 AEIT International Annual Conference*, 2017, pp. 1–6.
- [28] V. Radunović and M. Veinović, “Malware command and control over social media: Towards the server-less infrastructure,” *SJEE*, vol. 17, no. 3, pp. 357–375, 2020.
- [29] S. Miller and P. Smith, “Rise of legitimate services for backdoor command and control,” Anomali, Tech. Rep., 2017.

- [30] M. Botacin, H. Aghakhani, S. Ortolani, C. Kruegel, G. Vigna, D. Oliveira, P. L. D. Geus, and A. Grégio, “One size does not fit all: A longitudinal analysis of brazilian financial malware,” *ACM Trans. Priv. Secur.*, vol. 24, no. 2, Jan. 2021.
- [31] VirusTotal, *VirusTotal*, Jun. 2004. [Online]. Available: <https://www.virustotal.com/>.
- [32] J. Fuller, R. P. Kasturi, A. Sikder, H. Xu, B. Arik, V. Verma, E. Asdar, and B. Saltaformaggio, “C3PO: Large-Scale Study of Covert Monitoring of C&C Servers via Over-Permissioned Protocol Infiltration,” in *Proceedings of the 28th ACM Conference on Computer and Communications Security (CCS)*, Seoul, South Korea, Nov. 2021.
- [33] C. Rossow, D. Andriess, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, “Sok: P2pwned - modeling and evaluating the resilience of peer-to-peer botnets,” in *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2013, pp. 97–111.
- [34] B. B. Kang, E. Chan-Tin, C. P. Lee, J. Tyra, H. J. Kang, C. Nunnery, Z. Wadler, G. Sinclair, N. Hopper, D. Dagon, et al., “Towards Complete Node Enumeration in a Peer-to-peer Botnet,” in *Proceedings of the 4th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Sydney, Australia, Mar. 2009, pp. 23–34.
- [35] MITRE ATT&CK Web Service, <https://attack.mitre.org/techniques/T1102/>, [Accessed: 2021-04-21].
- [36] F. FKIE, *Malpedia: Free and Open Malware Reverse Engineering Resource offered by Fraunhofer FKIE*, Nov. 2021. [Online]. Available: <https://malpedia.caad.fkie.fraunhofer.de>.
- [37] MITRE ATT&CK, Web Service, Nov. 2021. [Online]. Available: <https://attack.mitre.org/techniques/T1102/>.

- [38] V. Chipounov, V. Kuznetsov, and G. Candea, “S2E: A Platform for In-vivo Multi-path Analysis of Software Systems,” *ACM SigPlan Notices*, vol. 46, no. 3, pp. 265–278, 2011.
- [39] Y. Hu, G. Huang, and P. Huang, “Automated reasoning and detection of specious configuration in large systems with symbolic execution,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 719–734.
- [40] A. Altinay, J. Nash, T. Kroes, P. Rajasekaran, D. Zhou, A. Dabrowski, D. Gens, Y. Na, S. Volckaert, C. Giuffrida, et al., “Binrec: Dynamic binary lifting and recompilation,” in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.
- [41] S. Walla and C. Rossow, “MALPITY: Automatic Identification and Exploitation of Tarpit Vulnerabilities in Malware,” in *Proceedings of the 4th European Symposium on Security and Privacy (EuroS&P)*, Stockholm, Sweden, Jun. 2019, pp. 590–605.
- [42] S. Sebastián and J. Caballero, “AVclass2: Massive Malware Tag Extraction from AV Labels,” in *Proceedings of the 36th Annual Computer Security Applications Conference (ACSAC)*, Virtual Conference, Dec. 2020, pp. 42–53.
- [43] *VirusTotal - Intelligence Search*, <https://developers.virustotal.com/reference/intelligence-search>, [Accessed: 2022-10-01].
- [44] O. Alrawi, C. Lever, K. Valakuzhy, K. Snow, F. Monroe, M. Antonakakis, et al., “The circle of life: A large-scale study of the iot malware lifecycle,” in *Proceedings of the 30th USENIX Security Symposium (Security)*, Virtual Conference, Aug. 2021.
- [45] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, “Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation,” in *Proceedings of the 2019 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2019.

- [46] *Cisco Talos Intelligence Group - Comprehensive Threat Intelligence*, <https://www.talosintelligence.com>, [Accessed: 2022-04-15].
- [47] M. A. Islam, A. B. M. A. A. Islam, and M. S. H. Anik, "Polygot: An approach towards reliable translation by name identification and memory optimization using semantic analysis," in *2017 4th International Conference on Networking, Systems and Security (NSysS)*, 2017, pp. 1–8.
- [48] K. Clark, M. Warnier, and F. M. Brazer, "The future of cloud-based botnets?" In *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER)*, Noordwijkerhout, Netherlands, May 2011.
- [49] X. Han, N. Kheir, and D. Balzarotti, "The role of cloud services in malicious software: Trends and insights," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2015, pp. 187–204.
- [50] D. Barroso, "Botnets - the silent threat," *European Network and Information Security Agency (ENISA)*, vol. 15, p. 171, 2007.
- [51] J. Zhang, C. Yang, Z. Xu, and G. Gu, "Poisonamplifier: A guided approach of discovering compromised websites through reversing search poisoning attacks," in *Proceedings of the 15th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Amsterdam, Netherlands, Sep. 2012.
- [52] Z. Li, S. Alrwais, Y. Xie, F. Yu, and X. Wang, "Finding the linchpins of the dark web: A study on topologically dedicated hosts on malicious web infrastructures," in *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2013.
- [53] S. Alrwais, K. Yuan, E. Alowaisheq, Z. Li, and X. Wang, "Understanding the dark side of domain parking," in *Proceedings of the 22th USENIX Security Symposium (Security)*, Washington, DC, Aug. 2013.

- [54] *Amazon EC2 - Secure Cloud Services*, <https://aws.amazon.com/ec2/>, [Accessed: 2022-10-01].
- [55] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis, “A Lustrum of Malware Network Communication: Evolution and Insights,” in *Proceedings of the 38th IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, May 2017, pp. 788–804.
- [56] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. C. Mitchell, “A Layered Architecture for Detecting Malicious Behaviors,” in *Proceedings of the 11th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Cambridge, Massachusetts, Sep. 2008, pp. 78–97.
- [57] E. Stinson and J. C. Mitchell, “Characterizing Bots’ Remote Control Behavior,” in *Proceedings of the Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, Lucerne, CH, Jul. 2007, pp. 89–108.
- [58] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer, “Behavior-based Spyware Detection,” in *Proceedings of the 15th USENIX Security Symposium (Security)*, Vancouver, Canada, Jul. 2006, p. 694.
- [59] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-y. Zhou, and X. Wang, “Effective and Efficient Malware Detection at the End Host,” in *Proceedings of the 18th USENIX Security Symposium (Security)*, vol. 4, Montreal, Canada, Aug. 2009, pp. 351–366.
- [60] G. J. Széles and A. Coleşa, “Malware Clustering Based on Called API During Runtime,” in *Proceedings of the International Workshop on Information and Operational Technology and Security (IOSec)*, Crete, GR, Sep. 2018, pp. 110–121.
- [61] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, “Copperdroid: Automatic reconstruction of android malware behaviors,” in *Proceedings of the 2015 Annual*

Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2015.

- [62] A. Lanzi, M. I. Sharif, and W. Lee, “K-tracer: A system for extracting kernel malware behavior,” in *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2009.
- [63] O. Alrawi, M. Ike, M. Pruett, R. P. Kasturi, S. Barua, T. Hirani, B. Hill, and B. Saltaformaggio, “Forecasting Malware Capabilities From Cyber Attack Memory Images,” in *Proceedings of the 30th USENIX Security Symposium (Security)*, Virtual Conference, Aug. 2021.
- [64] X. Deng and J. Mirkovic, “Malware Analysis Through High-level Behavior,” in *Proceedings of the 11th USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, Baltimore, MD, Aug. 2018.
- [65] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, “A View on Current Malware Behaviors,” in *Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, Boston, MA, Apr. 2009.
- [66] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, “Detecting android malware leveraging text semantics of network flows,” *IEEE Transactions on Information Forensics and Security (TIFS)*, 2018.
- [67] P. Godefroid, N. Klarlund, and K. Sen, “DART: Directed automated random testing,” in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Chicago, IL, Jun. 2005.
- [68] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler, “EXE: Automatically generating inputs of death,” *ACM Transactions on Information and System Security*, 2008.

- [69] C. Cadar, D. Dunbar, D. R. Engler, et al., “Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs,” in *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, San Diego, CA, Dec. 2008.
- [70] P. Godefroid, M. Y. Levin, D. A. Molnar, et al., “Automated whitebox fuzz testing,” in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2008.
- [71] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, et al., “Sok:(state of) the art of war: Offensive techniques in binary analysis,” in *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, May 2016.
- [72] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, “Unleashing Mayhem on Binary Code,” in *Proceedings of the 33rd IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2012, pp. 380–394.
- [73] L. Martignoni, S. McCamant, P. Poosankam, D. Song, and P. Maniatis, “Path-exploration lifting: Hi-fi tests for lo-fi emulators,” *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 337–348, Mar. 2012.
- [74] E. Bounimova, P. Godefroid, and D. Molnar, “Billions and billions of constraints: Whitebox fuzz testing in production,” in *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, May 2013.
- [75] I. Haller, A. Slowinska, M. Neugschwandtner, and H. Bos, “Dowsing for overflows: A guided fuzzer to find buffer boundary violations,” in *Proceedings of the 22th USENIX Security Symposium (Security)*, Washington, DC, Aug. 2013.
- [76] M. N. Alsaleh, J. Wei, E. Al-Shaer, and M. Ahmed, “Gextractor: Towards automated extraction of malware deception parameters,” in *Proceedings of the 8th Software Security, Protection, and Reverse Engineering Workshop*, 2018, pp. 1–12.

- [77] S. Y. Kim, S. Lee, I. Yun, W. Xu, B. Lee, Y. Yun, and T. Kim, “Cab-fuzz: Practical concolic testing techniques for cots operating systems,” in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 689–701.
- [78] *GitHub Terms of Service*, <https://docs.github.com/en/site-policy/github-terms/github-terms-of-service>, [Accessed: 2022-10-01].
- [79] *Report Abuse*, <https://support.github.com/contact/report-abuse>, [Accessed: 2022-10-01].
- [80] *Discord’s Terms of Service*, <https://discord.com/terms>, [Accessed: 2022-10-01].
- [81] *Discord’s Request Submission*, <https://dis.gd/request>, [Accessed: 2022-10-01].
- [82] *Pastebin.com Terms of Service*, https://pastebin.com/doc_terms_of_service, [Accessed: 2022-10-01].
- [83] *Pastebin Report*, <https://pastebin.com/contact>, [Accessed: 2022-10-01].
- [84] D. Brumley, C. Hartwig, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, D. Song, and H. Yin, “Bitscope: Automatically dissecting malicious binaries,” CS-07-133, School of Computer Science, Carnegie Mellon University, Tech. Rep., 2007.
- [85] Y. Li, Z. Su, L. Wang, and X. Li, “Steering symbolic execution to less traveled paths,” *ACM SigPlan Notices*, vol. 48, no. 10, pp. 19–32, 2013.
- [86] V. Kuznetsov, J. Kinder, S. Bucur, and G. Candea, “Efficient state merging in symbolic execution,” *Acm Sigplan Notices*, vol. 47, no. 6, pp. 193–204, 2012.
- [87] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, “Dynamic analysis of malicious code,” *Journal in Computer Virology*, vol. 2, no. 1, pp. 67–77, 2006.
- [88] A. Moser, C. Kruegel, and E. Kirda, “Exploring multiple execution paths for malware analysis,” in *2007 IEEE Symposium on Security and Privacy (SP’07)*, IEEE, 2007, pp. 231–245.

- [89] G. Jacob, R. Hund, C. Kruegel, and T. Holz, “JACKSTRAWS: Picking Command and Control Connections from Bot Traffic,” in *Proceedings of the 20th USENIX Security Symposium (Security)*, San Francisco, CA, Aug. 2011.
- [90] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou II, and D. Dagon, “Detecting Malware Domains at the Upper DNS Hierarchy,” in *Proceedings of the USENIX Security Symposium*, San Francisco, CA, 2011, pp. 25–40.
- [91] J. Menn, *Court orders seizure of ransomware botnet controls as U.S. election nears*, Sep. 2020. [Online]. Available: <https://www.reuters.com/article/us-uselection-cyber-botnet/court-orders-seizure-of-ransomware-botnet-controls-as-u-s-election-nears-idUSKBN26X1G2>.
- [92] C. Page, *Rsocks, A popular proxy service, was just seized by the DOJ*, Jun. 2022. [Online]. Available: <https://techcrunch.com/2022/06/17/rsocks-proxy-seized-justice-department/>.
- [93] R. Vogt, J. Aycock, and M. J. Jacobson Jr, “Army of Botnets,” in *Proceedings of the 14th Annual Network and Distributed System Security Symposium*, San Diego, CA, 2007.
- [94] Z. Xu, A. Nappa, R. Baykov, G. Yang, J. Caballero, and G. Gu, “Autoprobe: Towards Automatic Active Malicious Server Probing Using Dynamic Binary Analysis,” in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, Scottsdale, AZ, Nov. 2014, pp. 179–190.
- [95] A. Nappa, Z. Xu, M. Z. Rafique, J. Caballero, and G. Gu, “Cyberprobe: Towards Internet-scale Active Detection of Malicious Servers,” in *Proceedings of the 2014 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2014, pp. 1–15.

- [96] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee, “Bothunter: Detecting malware infection through ids-driven dialog correlation,” in *16th USENIX Security Symposium (USENIX Security 07)*, Boston, MA, 2007, pp. 113–128.
- [97] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, “BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection,” in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2012)*, Nice, France: Association for Computing Machinery, Nov. 2012, pp. 349–360.
- [98] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, “Disclosure: Detecting Botnet Command and Control Servers through Large-Scale Netflow Analysis,” in *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC)*, Orlando, FL, USA: ACM, Dec. 2012, pp. 129–138.
- [99] R. Zhang, M. Yao, H. Xu, O. Alrawi, J. Park, and B. Saltaformaggio, “Hitchhiking Vaccine: Enhancing Botnet Remediation With Remote Code Deployment Reuse,” in *Proceedings of the Network and Distributed System Security Symposium*, 2025.
- [100] P. Arntz, *Analyzing malware by API calls*, Oct. 2017. [Online]. Available: <https://blog.malwarebytes.com/threat-analysis/2017/10/analyzing-malware-by-api-calls/>.
- [101] E. Research, *Casbaneiro: Dangerous Cooking with a Secret Ingredient*, Oct. 2019. [Online]. Available: <https://www.welivesecurity.com/2019/10/03/casbaneiro-trojan-dangerous-cooking/>.
- [102] Securelist, *The Dropping Elephant - Aggressive Cyber-Espionage in the Asian Region*, Mar. 2022. [Online]. Available: <https://securelist.com/the-dropping-elephant-actor/75328/>.

- [103] A. Team, *Donot Team Leverages New Framework*, Mar. 2018. [Online]. Available: <https://www.netscout.com/blog/asert/donot-team-leverages-new-modular-malware-framework-south-asia>.
- [104] S. Pletinckx, C. Trap, and C. Doerr, “Malware Coordination Using the Blockchain: An Analysis of the Cerber Ransomware,” in *Proceedings of the 6th IEEE Conference on Communications and Network Security (CNS)*, Beijing, China: IEEE, May 2018, pp. 1–9.
- [105] G. Gomez, P. Moreno-Sanchez, and J. Caballero, “Watch Your Back: Identifying Cybercrime Financial Relationships in Bitcoin through Back-and-Forth Exploration,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, Los Angeles, CA, 2022, pp. 1291–1305.
- [106] T. Taniguchi, H. Griffioen, and C. Doerr, “Analysis and takeover of the bitcoin-coordinated pony malware,” ASIACCS21.
- [107] T. Nelms, R. Perdisci, and M. Ahamad, “Execscent: Mining for new C&C Domains in Live Networks with Adaptive Control Protocol Templates,” in *Proceedings of the 22th USENIX Security Symposium (Security)*, Washington, DC, Aug. 2013.
- [108] X. Liao, S. A. Alrwais, K. Yuan, L. Xing, X. Wang, S. Hao, and R. A. Beyah, “Lurking Malice in the Cloud: Understanding and Detecting Cloud Repository as a Malicious Service,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.
- [109] G. Gu, J. Zhang, and W. Lee, “BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic,” in *Proceedings of the 15th Annual Network and Distributed System Security Symposium*, 2008.
- [110] I. Ghafir, V. Prenosil, M. Hammoudeh, T. Baker, S. Jabbar, S. Khalid, and S. F. Jaf, “BotDet: A System for Real Time Botnet Command and Control Traffic Detection,” *IEEE Access*, vol. 6, pp. 38 947–38 958, Jul. 2018.

- [111] Z. Huang, J. Huang, and T. Zang, “Leopard: Understanding the Threat of Blockchain Domain Name Based Malware,” in *Passive and Active Measurement: 21st International Conference, PAM 2020*, Eugene, OR, USA: Springer, Mar. 2020, pp. 3–19.
- [112] Netresec, *Publicly Available PCAP Files*, 2023. [Online]. Available: <https://www.netresec.com/?page=PcapFiles>.
- [113] WordPress, *WordPress.com: Build a Site, Sell Your Stuff, Start a Blog & More*, 2024. [Online]. Available: <https://wordpress.com/>.
- [114] FIRST, *FIRST is the global Forum of Incident Response and Security Teams*, Oct. 2015. [Online]. Available: <https://www.first.org>.
- [115] CYFIRMA, *Cyber Research on the Malicious Use of Discord*, Sep. 2022. [Online]. Available: <https://www.cyfirma.com/research/cyber-research-on-the-malicious-use-of-discord>.
- [116] N. Todorovic and A. Chaudhuri, *Using AI to help organizations detect and report child sexual abuse material online*, 2023. [Online]. Available: <https://blog.google/around-the-globe/google-europe/using-ai-help-organizations-detect-and-report-child-sexual-abuse-material-online/>.
- [117] T. Intercept, *The Feds Are Coming For "Extremist" Gamers*, Mar. 2024. [Online]. Available: <https://theintercept.com/2024/03/09/fbi-dhs-gamers-extremism-violence/>.
- [118] T. Micro, *Trend Micro Collaborated with Interpol in Cracking Down Grandoreiro Banking Trojan*, Jul. 2024. [Online]. Available: <https://www.trendmicro.com/en-fi/research/24/d/trend-micro-collaborated-with-interpol-in-cracking-down-grandore.html>.

- [119] L. Bermejo and J. Soares, *Lazarus Targets Latin American Financial Companies*, Nov. 2018. [Online]. Available: https://www.trendmicro.com/en_us/research/18/k/lazarus-continues-heists-mounts-attacks-on-financial-organizations-in-latin-america.html.
- [120] FireEye, *Double Dragon - APT41, a Dual Espionage and Cyber Crime Operation*, Aug. 2019. [Online]. Available: <https://content.fireeye.com/apt-41/rpt-apt41>.
- [121] Forcepoint, *Monsoon – Analysis of an APT Campaign*, Mar. 2022. [Online]. Available:
<https://www.forcepoint.com/sites/default/files/resources/files/forcepoint-security-labs-monsoon-analysis-report.pdf>.
- [122] T. Micro, *Understanding the Patchwork Cyberespionage Group*, Mar. 2022. [Online]. Available: <https://documents.trendmicro.com/assets/tech-brief-untangling-the-patchwork-cyberespionage-group.pdf>.
- [123] C. T. U. R. Team, *Bronze Butler Targets Japanese Enterprises*, Oct. 2017. [Online]. Available: <https://www.secureworks.com/research/bronze-butler-targets-japanese-businesses>.
- [124] M. Labs, *Explained: Spora ransomware*, Mar. 2017. [Online]. Available: <https://www.malwarebytes.com/blog/news/2017/03/spora-ransomware>.
- [125] T. Micro, *Operation Endtrade: Tick's Multi-Stage Backdoors for Attacking Industries and Stealing Classified Data*, Nov. 2022. [Online]. Available: <https://documents.trendmicro.com/assets/pdf/Operation-ENDTRADE-TICK-s-Multi-Stage-Backdoors-for-Attacking-Industries-and-Stealing-Classified-Data.pdf>.
- [126] A. Cherepanov, *Analysis of TeleBots' cunning backdoor*, Jul. 2017. [Online]. Available: <https://www.welivesecurity.com/2017/07/04/analysis-of-telebots-cunning-backdoor/>.

- [127] Z. Hromcová and A. Cherepanov, *Invisimole: The Hidden Part of the Story Unearthing Invisimole's Espionage Toolset and Strategic Cooperations*, Jun. 2020. [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/2020/06/ESET_InvisiMole.pdf.
- [128] C. Research, *Pony's C&C Servers Hidden Inside the Bitcoin Blockchain*, Dec. 2019. [Online]. Available: <https://research.checkpoint.com/2019/ponys-cc-servers-hidden-inside-the-bitcoin-blockchain/>.
- [129] ESET, *Operation Ghost. The Dukes aren't back — they never left*, Oct. 2019. [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/2019/10/ESET_Operation_Ghost_Dukes.pdf.
- [130] FireEye, *Multigrain - Point of Sale Attackers Make an Unhealthy Addition to the Pantry*, Apr. 2016. [Online]. Available: https://www.fireeye.com/blog/threat-research/2016/04/multigrain_pointo.html.
- [131] V. Ventura, *Gustuff Banking Botnet Targets Australia*, Apr. 2019. [Online]. Available: <https://blog.talosintelligence.com/2019/04/gustuff-targets-australia.html>.
- [132] J. C. Chen, K. Lu, J. Horejsi, and G. Chen, *Biopass RAT: New Malware Sniffs Victims via Live Streaming*, Jul. 2021. [Online]. Available: https://www.trendmicro.com/en_us/research/21/g/biopass-rat-new-malware-sniffs-victims-via-live-streaming.html.
- [133] T. Micro, *Malware Campaign Targets South Korean Banks*, Nov. 2021. [Online]. Available: <https://blog.trendmicro.com/trendlabs-security-intelligence/malware-campaign-targets-south-korean-banks-uses-pinterest-as-cc-channel/>.
- [134] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines," in *29th USENIX Security Symposium (USENIX Security 20)*, Boston, MA, 2020, pp. 1767–1784.

- [135] Z. Liu, Z. Zhang, Y. Cao, Z. Xi, S. Jing, and H. La Roche, “Towards a secure zero-rating framework with three parties,” in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, 2018, pp. 1885–1902.
- [136] S. Kleber and F. Kargl, “Poster: Network message field type recognition,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.
- [137] Abuse.ch, *URLHaus*, 2023. [Online]. Available: <https://urlhaus.abuse.ch/>.
- [138] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, “From Throw-away Traffic to Bots: Detecting the Rise of DGA-based Malware,” in *Proceedings of the 21st USENIX Security Symposium (Security)*, Bellevue, WA, Aug. 2012, pp. 491–506.
- [139] V. Le Pochat, S. Maroofi, T. Van Goethem, D. Preuveneers, A. Duda, W. Joosen, M. Korczyński, et al., “A Practical Approach for Taking Down Avalanche Botnets Under Real-World Constraints,” in *Proceedings of the 2020 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2020.
- [140] P. Kotzias, L. Bilge, and J. Caballero, “Measuring PUP Prevalence and PUP Distribution through Pay-Per-Install Services,” in *Proceedings of the 25th USENIX Security Symposium (Security)*, Austin, TX, Aug. 2016, pp. 739–756.
- [141] X. Mi, X. Feng, X. Liao, B. Liu, X. Wang, F. Qian, Z. Li, S. Alrwais, L. Sun, and Y. Liu, “Resident Evil: Understanding Residential IP Proxy as a Dark Service,” in *Proceedings of the 40th IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2019, pp. 1185–1201.
- [142] D. Kim, B. J. Kwon, K. Kozák, C. Gates, and T. Dumitras, “The Broken Shield: Measuring Revocation Effectiveness in the Windows Code-Signing PKI,” in *Proceedings of the 27th USENIX Security Symposium (Security)*, Baltimore, MD, Aug. 2018, pp. 851–868.

- [143] Stanford.Edu, *A System for Detecting Software Similarity*, Nov. 2022. [Online]. Available: <https://theory.stanford.edu/~aiken/moss/>.
- [144] D. Shamah, *How malware writers' laziness is helping one startup predict attacks before they even happen*, Oct. 2014. [Online]. Available: <https://www.zdnet.com/article/how-malware-writers-laziness-is-helping-one-startup-predict-attacks-before-they-even-happen/>.
- [145] M. I. ATT&CK, *Attack Matrix for Enterprise*, Nov. 2021. [Online]. Available: <https://attack.mitre.org/>.
- [146] A. Küchler, A. Mantovani, Y. Han, L. Bilge, and D. Balzarotti, "Does Every Second Count? Time-Based Evolution of Malware Behavior in Sandboxes," in *Proceedings of the 2021 Annual Network and Distributed System Security Symposium (NDSS)*, Virtual Conference, Feb. 2021.
- [147] L. Maffia, D. Nisi, P. Kotzias, G. Lagorio, S. Aonzo, and D. Balzarotti, "Longitudinal Study of the Prevalence of Malware Evasive Techniques," *arXiv preprint arXiv:2112.11289*, 2021.
- [148] N. Galloro, M. Polino, M. Carminati, A. Continella, and S. Zanero, "A Systematical and Longitudinal Study of Evasive Behaviors in Windows Malware," *COSE22*, vol. 113,
- [149] T. Blazytko, M. Contag, C. Aschermann, and T. Holz, "Syntia: Synthesizing the Semantics of Obfuscated Code," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017.
- [150] D. Xu, J. Ming, and D. Wu, "Cryptographic Function Detection in Obfuscated Binaries via Bit-Precise Symbolic Loop Mapping," in *Proceedings of the 38th IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, May 2017.

- [151] J. Calvet, J. M. Fernandez, and J.-Y. Marion, “Aligot: cryptographic function identification in obfuscated binary programs,” in *Proceedings of the 2012 ACM SIGSAC Conference on Computer and Communications Security*, 2012.
- [152] M. Torkashvan and H. Haghighi, “CB2C: A Cloud-Based Botnet Command and Control,” *Indian Journal of Science and Technology*, vol. 8, no. 32, pp. 1–8, Nov. 2015.
- [153] W. Lu, M. Miller, and L. Xue, “Detecting Command and Control Channel of Botnets in Cloud,” in *Proceedings of the 1st International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments (ISDDC)*, Vancouver, Canada: ACM, Oct. 2017, pp. 1–8.
- [154] S. Zhao, P. P. C. Lee, J. C. S. Lui, X. Guan, X. Ma, and J. Tao, “Cloud-Based Push-Styled Mobile Botnets: A Case Study of Exploiting the Cloud to Device Messaging Service,” in *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC 2012)*, Orlando, Florida, USA: ACM, Dec. 2012, pp. 393–402.
- [155] R. Zhang, R. P. Sridhar, M. Yao, Z. Yang, D. Oygenblik, H. Xu, V. Dave, C. Herley, P. England, and B. Saltaformaggio, “Identifying Incoherent Search Sessions: Search Click Fraud Remediation Under Real-World Constraints,” in *2025 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, 2025.
- [156] R. Duan, O. Alrawi, R. P. Kasturi, R. Elder, B. Saltaformaggio, and W. Lee, “Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages,” in *Proceedings of the 28th Annual Network and Distributed System Security Symposium (NDSS)*, Internet Society, 2021.
- [157] M. Gimenez-Aguilar, J. M. de Fuentes, and L. Gonzalez-Manzano, “Malicious uses of Blockchains by Malware: From the Analysis to Smart-Zephyrus,” *International Journal of Information Security*, vol. 22, no. 1, pp. 1–36, Feb. 2023.

- [158] J. Caballero, N. M. Johnson, S. McCamant, and D. Song, “Binary Code Extraction and Interface Identification for Security Applications,” in *Proceedings of the 17th Annual Network and Distributed System Security Symposium*, San Diego, CA, 2010, pp. 319–334.
- [159] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda, “Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries,” in *Proceedings of the 31st IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 2010, pp. 29–44.
- [160] P. Shirani, L. Wang, and M. Debbabi, “Binshape: Scalable and Robust Binary Library Function Identification using Function Shape,” in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2017)*, Bonn, Germany: Springer, Jul. 2017, pp. 227–247.
- [161] J. Ming, D. Xu, Y. Jiang, and D. Wu, “BinSim: Trace-based Semantic Binary Diffing via System Call Sliced Segment Equivalence Checking,” in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, Canada, 2017, pp. 707–724.
- [162] Y. David, N. Partush, and E. Yahav, “Statistical Similarity of Binaries,” in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Santa Barbara, CA, USA: Association for Computing Machinery, Jun. 2016, pp. 266–280.
- [163] L. Luo, J. Ming, D. Wu, P. Liu, and S. Zhu, “Semantics-Based Obfuscation-Resilient Binary Code Similarity Comparison With Applications to Software Plagiarism Detection,” in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*, Hong Kong, China: Association for Computing Machinery, Nov. 2014, pp. 389–400.
- [164] M. Chandramohan, Y. Xue, Z. Xu, Y. Liu, C. Y. Cho, and H. B. K. Tan, “Bingo: Cross-Architecture Cross-OS Binary Search,” in *Proceedings of the 24th ACM*

- SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*, Seattle, WA, USA: ACM, Nov. 2016, pp. 678–689.
- [165] J. Pewny, B. Garmany, R. Gawlik, C. Rossow, and T. Holz, “Cross-Architecture Bug Search in Binary Executables,” in *2015 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, 2015, pp. 715–730.
- [166] S. Wang and D. Wu, “In-memory Fuzzing for Binary Code Similarity Analysis,” in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*, Urbana, IL, USA: IEEE, Oct. 2017, pp. 319–324.
- [167] T. Bao, J. Burket, M. Woo, R. Turner, and D. Brumley, “BYTEWEIGHT: Learning to Recognize Functions in Binary Code,” in *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA, 2014, pp. 59–74.
- [168] C. Cadar and D. Engler, “Execution Generated Test Cases: How to Make Systems Code Crash Itself,” in *Proceedings of the International SPIN Workshop on Model Checking of Software*, Springer, San Francisco, CA, USA, Aug. 2005, pp. 2–23.
- [169] V. Chipounov, V. Georgescu, C. Zamfir, and G. Candea, “Selective Symbolic Execution,” in *Proceedings of the 5th Workshop on Hot Topics in System Dependability (HotDep)*, Estoril, Portugal, Jun. 2009.
- [170] J. C. King, “Symbolic Execution and Program Testing,” 7, vol. 19, ACM, 1976, pp. 385–394.
- [171] R. S. Boyer, B. Elspas, and K. N. Levitt, “SELECT — A Formal System for Testing and Debugging Programs by Symbolic Execution,” 6, vol. 10, ACM, 1975, pp. 234–245.
- [172] L. A. Clarke, “A System to Generate Test Data and Symbolically Execute Programs,” 3, IEEE, 1976, pp. 215–222.

- [173] F. Peng, Z. Deng, X. Zhang, D. Xu, Z. Lin, and Z. Su, “X-force: Force-executing Binary Programs for Security Applications,” in *Proceedings of the 23rd USENIX Security Symposium (Security)*, San Diego, CA, Aug. 2014, pp. 829–844.
- [174] K. Kim, I. L. Kim, C. H. Kim, Y. Kwon, Y. Zheng, X. Zhang, and D. Xu, “J-force: Forced Execution on Javascript,” in *Proceedings of the 26th International World Wide Web Conference (WWW)*, Perth, Australia, Apr. 2017, pp. 897–906.
- [175] A. Naderi-Afooshteh, Y. Kwon, A. Nguyen-Tuong, A. Razmjoo-Qalaei, M.-R. Zamiri-Gourabi, and J. W. Davidson, “MalMax: Multi-Aspect Execution for Automated Dynamic Web Server Malware Analysis,” in *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS)*, London, UK, Nov. 2019, pp. 1849–1866.
- [176] H. Xu, M. Yao, R. Zhang, M. M. Dawoud, J. Park, and B. Saltaformaggio, “DVa: Extracting Victims and Abuse Vectors from Android Accessibility Malware,” in *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA, 2024, pp. 6179–6196.
- [177] E. J. Schwartz, T. Avgerinos, and D. Brumley, “All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (But Might Have Been Afraid to Ask),” in *Proceedings of the IEEE Symposium on Security and Privacy (IEEE S&P)*, Oakland, CA, 2010, pp. 317–331.
- [178] M. G. Kang, S. McCamant, P. Poosankam, and D. Song, “Dta++: Dynamic Taint Analysis With Targeted Control-Flow Propagation,” in *Proceedings of the 18th Annual Network and Distributed System Security Symposium*, San Diego, CA, 2011, pp. 111–126.
- [179] J. Newsome and D. X. Song, “Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software,” in

Proceedings of the 12th Annual Network and Distributed System Security Symposium, San Diego, CA, 2005, pp. 1–15.

- [180] J. Clause, W. Li, and A. Orso, “Dytan: A Generic Dynamic Taint Analysis Framework,” in *Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA)*, London, United Kingdom: ACM, Jul. 2007, pp. 196–206.
- [181] *A Crypto Scam Is Born Every Four Minutes, Report Finds*, <https://www.wsj.com/livecoverage/stock-market-news-today-10-27-2022-us-economy-gdp-q3/card/a-crypto-scam-is-born-every-four-minutes-report-finds-EXvZpYofaSx3mgA0QFf1>, [Accessed: 2023-12-03].
- [182] *Most Crypto Scams on BNB Chain, Solidus Labs Says*, <https://www.coindesk.com/business/2022/10/27/most-crypto-scams-on-bnb-chain-solidus-labs-says/>, [Accessed: 2023-12-03].
- [183] *Ethereum (ETH) Blockchain Explorer - Etherscan*, <https://etherscan.io/>, [Accessed: 2023-12-03].
- [184] S. Yu, J. Jin, Y. Xie, J. Shen, and Q. Xuan, “Ponzi scheme detection in ethereum transaction network,” in *Proceedings of the 3rd International Conference on Blockchain and Trustworthy Systems (BlockSys)*, Guangzhou, China, Aug. 2021.
- [185] Y. Zhang, W. Yu, Z. Li, S. Raza, and H. Cao, “Detecting ethereum Ponzi schemes based on improved lightgbm algorithm,” *IEEE Transactions on Computational Social Systems*, vol. 9, no. 2, pp. 624–637, Jul. 2021.
- [186] X. Shen, S. Jiang, and L. Zhang, “Mining bytecode features of smart contracts to detect Ponzi scheme on blockchain,” *Computer Modeling in Engineering & Sciences*, vol. 127, no. 3, pp. 1069–1085, Jan. 2021.

- [187] Y. Chen, H. Dai, X. Yu, W. Hu, Z. Xie, and C. Tan, “Improving Ponzi scheme contract detection using multi-channel textcnn and transformer,” *Sensors*, vol. 21, no. 19, p. 6417, Sep. 2021.
- [188] H. Zhang, J. Yu, B. Yan, M. Jing, and J. Zhao, “Security on ethereum: Ponzi scheme detection in smart contract,” in *Proceedings of the 16th International Conference on Algorithmic Applications in Management (AAIM)*, Guangzhou, China, Aug. 2022.
- [189] Z. Zheng, W. Chen, Z. Zhong, Z. Chen, and Y. Lu, “Securing the ethereum from smart Ponzi schemes: Identification using static features,” *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, 130:1–130:28, 2023.
- [190] J. Peng and G. Xiao, “Detection of smart Ponzi schemes using opcode,” in *Proceedings of the 2nd International Conference on Blockchain and Trustworthy Systems (BlockSys)*, Dali, China, Aug. 2020.
- [191] Y. Zhang, S. Kang, W. Dai, S. Chen, and J. Zhu, “Code will speak: Early detection of Ponzi smart contracts on ethereum,” in *Proceedings of 2021 IEEE International Conference on Services Computing (SCC)*, Chicago, IL, Sep. 2021.
- [192] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, and Y. Zhou, “Detecting Ponzi schemes on ethereum: Towards healthier blockchain technology,” in *Proceedings of the 27th International World Wide Web Conference (WWW)*, Lyon, France, Apr. 2018.
- [193] Y. Lou, Y. Zhang, and S. Chen, “Ponzi contracts detection based on improved convolutional neural network,” in *Proceedings of the 2020 IEEE International Conference on Services Computing (SCC)*, Beijing, China, Mar. 2020.
- [194] W. Sun, G. Xu, Z. Yang, and Z. Chen, “Early detection of smart Ponzi scheme contracts based on behavior forest similarity,” in *Proceedings of the 20th International Conference on Software Quality, Reliability and Security (QRS)*, Macau, China, Dec. 2020.

- [195] L. Liu, W.-T. Tsai, M. Z. A. Bhuiyan, H. Peng, and M. Liu, "Blockchain-enabled fraud discovery through abnormal smart contract detection on ethereum," *Future Generation Computer Systems*, vol. 128, pp. 158–166, Mar. 2022.
- [196] T. Ashfaq, R. Khalid, A. S. Yahaya, S. Aslam, A. T. Azar, S. Alsafari, and I. A. Hameed, "A machine learning and blockchain based efficient fraud detection mechanism," *Sensors*, vol. 22, no. 19, p. 7162, Sep. 2022.
- [197] H. Hu, Q. Bai, and Y. Xu, "SCSGuard: Deep scam detection for ethereum smart contracts," in *Proceedings of IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Virtual Conference, May 2022.
- [198] R. M. Aziz, R. Mahto, K. Goel, A. Das, P. Kumar, and A. Saxena, "Modified genetic algorithm with deep learning for fraud transactions of ethereum smart contract," *Applied Sciences*, vol. 13, no. 2, p. 697, Jan. 2023.
- [199] *Wikipedia. Uniform Electronic Transactions Act*, https://en.wikipedia.org/wiki/Uniform_Electronic_Transactions_Act, [Accessed: 2023-11-13].
- [200] *Wikipedia. Electronic Signatures in Global and National Commerce Act*, https://en.wikipedia.org/wiki/Electronic_Signatures_in_Global_and_National_Commerce_Act, [Accessed: 2023-11-13].
- [201] *FBI Identifies Cryptocurrency Funds Stolen by DPRK*, <https://www.fbi.gov/news/press-releases/fbi-identifies-cryptocurrency-funds-stolen-by-dprk>, [Accessed: 2023-12-03].
- [202] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS)*, Vienna, Austria, Oct. 2016.
- [203] J. Krupp and C. Rossow, "teEther: Gnawing at ethereum to automatically exploit smart contracts," in *Proceedings of the 27th USENIX Security Symposium (Security)*, Baltimore, MD, Aug. 2018.

- [204] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, “Zeus: Analyzing safety of smart contracts,” in *Proceedings of the 2018 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2018.
- [205] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, “Finding the greedy, prodigal, and suicidal contracts at scale,” in *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*, San Juan, PR, Dec. 2018.
- [206] F. Ma, Z. Xu, M. Ren, Z. Yin, Y. Chen, L. Qiao, B. Gu, H. Li, Y. Jiang, and J. Sun, “Pluto: Exposing vulnerabilities in inter-contract scenarios,” *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4380–4396, Oct. 2021.
- [207] J. Frank, C. Aschermann, and T. Holz, “ETHBMC: A bounded model checker for smart contracts,” in *Proceedings of the 29th USENIX Security Symposium (Security)*, Virtual Conference, Aug. 2020.
- [208] P. Bose, D. Das, Y. Chen, Y. Feng, C. Kruegel, and G. Vigna, “SAILFISH: Vetting smart contract state-inconsistency bugs in seconds,” in *Proceedings of the 43rd IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2022.
- [209] *FBI - Federal Bureau of Investigation*, <https://www.fbi.gov/>, [Accessed: 2023-04-06].
- [210] *Wikipedia. SHA-3*, <https://en.wikipedia.org/wiki/SHA-3>, [Accessed: 2023-04-06].
- [211] *Crypterium - BitcoinWiki*, <https://en.bitcoinwiki.org/wiki/Crypterium>, [Accessed: 2023-04-06].
- [212] *Tether Assists US Government in Seizing \$1.4 Million From Unhosted Wallet*, <https://news.bitcoin.com/tether-assists-us-government-in-seizing-1-4-million-from-unhosted-wallet/>, [Accessed: 2024-03-14].
- [213] *Bleeping Computer. Ethereum feature abused to steal \$60 million from 99K victims*, <https://www.bleepingcomputer.com/news/security/ethereum-feature-abused-to-steal-60-million-from-99k-victims/>, [Accessed: 2023-11-13].

- [214] *GitHub. evm-labels*, <https://github.com/dawsbot/evm-labels>, [Accessed: 2023-11-13].
- [215] *Wikipedia. Ethereum*, <https://en.wikipedia.org/wiki/Ethereum>.
- [216] *Consensys - Mythril*, <https://github.com/Consensys/mythril>, [Accessed: 2023-04-06].
- [217] *GitHub. reth*, <https://github.com/paradigmxyz/reth>, [Accessed: 2023-11-13].
- [218] *GitHub. Panoramix*, <https://github.com/palkeo/panoramix>, [Accessed: 2023-11-13].
- [219] *Dedaub. Smart Contract Security Analysis*, <https://dedaub.com/>, [Accessed: 2023-11-13].
- [220] *Bitcoinwiki. ERC20*, <https://en.bitcoinwiki.org/wiki/ERC20>, [Accessed: 2023-11-13].
- [221] *Wikipedia. Non-fungible token*, https://en.wikipedia.org/wiki/Non-fungible_token, [Accessed: 2023-11-13].
- [222] *Binance Smart Chain*, <https://www.bnbchain.org/en/bnb-smart-chain>, [Accessed: 2023-11-13].
- [223] *Solana*, <https://solana.com/>, [Accessed: 2023-11-13].
- [224] B. He, Y. Chen, Z. Chen, X. Hu, Y. Hu, L. Wu, R. Chang, H. Wang, and Y. Zhou, “TxPhishScope: Towards detecting and understanding transaction-based phishing on ethereum,” in *Proceedings of the 30th ACM Conference on Computer and Communications Security (CCS)*, Copenhagen, Denmark, Nov. 2023.
- [225] W. Chen, X. Guo, Z. Chen, Z. Zheng, and Y. Lu, “Phishing scam detection on ethereum: Towards financial security for blockchain ecosystem.,” in *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, Yokohama, Japan, Jan. 2021.

- [226] S. Li, G. Gou, C. Liu, C. Hou, Z. Li, and G. Xiong, “TTAGN: Temporal transaction aggregation graph network for ethereum phishing scams detection,” in *Proceedings of the 31st International World Wide Web Conference (WWW)*, Virtual Conference, Apr. 2022.
- [227] P. Xia, H. Wang, B. Gao, W. Su, Z. Yu, X. Luo, C. Zhang, X. Xiao, and G. Xu, “Trade or trick? detecting and characterizing scam tokens on uniswap decentralized exchange,” in *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, vol. 5, Dec. 2021, pp. 1–26.
- [228] T. Chen, R. Cao, T. Li, X. Luo, G. Gu, Y. Zhang, Z. Liao, H. Zhu, G. Chen, Z. He, et al., “SODA: A generic online detection framework for smart contracts.,” in *Proceedings of the 2020 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2020.
- [229] C. Sendner, H. Chen, H. Fereidooni, L. Petzi, J. König, J. Stang, A. Dmitrienko, A.-R. Sadeghi, and F. Koushanfar, “Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning,” in *Proceedings of the 2023 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2023.
- [230] M. Eshghie, C. Artho, and D. Gurov, “Dynamic vulnerability detection on smart contracts using machine learning,” in *Proceedings of the 25th International Conference on Evaluation and Assessment in software engineering (EASE)*, Trondheim, Norway, Jun. 2021.
- [231] S. Gopali, Z. A. Khan, B. Chhetri, B. Karki, and A. S. Namin, “Vulnerability detection in smart contracts using deep learning,” in *Proceedings of 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, Jun. 2022.
- [232] H. Wu, Z. Zhang, S. Wang, Y. Lei, B. Lin, Y. Qin, H. Zhang, and X. Mao, “Peculiar: Smart contract vulnerability detection based on crucial data flow graph and

- pre-training techniques,” in *Proceedings of 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, Wuhan, China, Oct. 2021.
- [233] S. Cui, G. Zhao, Y. Gao, T. Tavu, and J. Huang, “VRust: Automated vulnerability detection for solana smart contracts,” in *Proceedings of the 29th ACM Conference on Computer and Communications Security (CCS)*, Los Angeles, CA, Nov. 2022.
- [234] Y. Wang, J. Zhao, Y. Zhang, X. Hei, and L. Zhu, “Smart contract symbol execution vulnerability detection method based on CFG path pruning,” in *Proceedings of the 5th ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI)*, Melbourne, VIC, Australia, Sep. 2023.
- [235] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, and X. Wang, “Combining graph neural networks with expert knowledge for smart contract vulnerability detection,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 1296–1310, Feb. 2021.
- [236] F. Gritti, N. Ruaro, R. McLaughlin, P. Bose, D. Das, I. Grishchenko, C. Kruegel, and G. Vigna, “Confusum contractum: Confused deputy vulnerabilities in ethereum smart contracts,” in *Proceedings of the 32nd USENIX Security Symposium (Security)*, Anaheim, CA, Aug. 2023.
- [237] C. Lab, *Defensive Evasion APIs*, Apr. 2025. [Online]. Available: https://github.com/CyFI-Lab-Public/VADER/blob/main/appendix/defensive_evasion_apis.pdf.
- [238] C. Lab, *Baseline Comparison For De-Manipulation Algorithm C/C++ Source Code Via Moss*, Apr. 2025. [Online]. Available: <https://github.com/CyFI-Lab-Public/VADER/blob/main/appendix/moss.pdf>.
- [239] C. Lab, *Baseline Comparison For Decoding Algorithm Similarity Via Symbolic Expressions*, Apr. 2025. [Online]. Available: https://github.com/CyFI-Lab-Public/VADER/blob/main/appendix/baseline_comparison.pdf.

- [240] C. Lab, *Identified Dead Drops*, Apr. 2025. [Online]. Available: https://github.com/CyFI-Lab-Public/VADER/blob/main/appendix/identified_accounts.pdf.

VITA

Mingxuan Yao was born to parents Kexia and Duochang. He resides in Atlanta, Georgia, where he recently completed his doctoral studies in cybersecurity and cyber forensics at the Georgia Institute of Technology. His academic research has focused on malware analysis, Ethereum smart contract security, and forensic evidence collection from artificial intelligence systems, leading to the development of core research projects including MARSEA, VADER, and COCO.

Outside of his academic and professional pursuits, Mingxuan enjoys managing self-hosted home laboratory services, exploring precision cooking techniques, and spending quiet time with his Golden Retriever, Coco, and his cat, Yuki.