

**THE BOT REVEALS ITS MASTER: EXPOSING AND INFILTRATING BOTNET  
COMMAND AND CONTROL SERVERS VIA MALWARE LOGIC REUSE**

A Dissertation  
Presented to  
The Academic Faculty

By

Jonathan D. Fuller

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 2022

© Jonathan D. Fuller 2022

**THE BOT REVEALS ITS MASTER: EXPOSING AND INFILTRATING BOTNET  
COMMAND AND CONTROL SERVERS VIA MALWARE LOGIC REUSE**

Thesis committee:

Dr. Brendan Saltaformaggio, Advisor  
School of Cybersecurity and Privacy and  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Dr. Frank Li  
School of Cybersecurity and Privacy and  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Dr. Mustaque Ahamad  
School of Cybersecurity and Privacy and  
School of Computer Science  
*Georgia Institute of Technology*

Dr. Jon Lindsay  
School of Cybersecurity and Privacy and  
Sam Nunn School of International Affairs  
*Georgia Institute of Technology*

Dr. Stephen Hamilton  
Army Cyber Institute and United States  
Military Academy  
*United States Army*

Date approved: April 25, 2022

*For what will it profit a man if he gains the whole world and forfeits his soul? Or what shall a man give in return for his soul?*

*Matthew 16:26*

To my dearest Leigh,

I loved you then.

I love you now.

I will love you always.

## ACKNOWLEDGMENTS

My Ph.D. pursuit was more challenging than I imagined. I often compare it to a marathon at sprint pace. Thankfully, God has been gracious to me through the support of my research lab, my friends, my family, and most importantly, his son, Jesus. God's mercy in my life is unmerited, and I am beholden to follow him all the days of my life.

I am grateful to my advisor, Dr. Brendan Saltaformaggio. He has been my mentor, advocate and has provided invaluable encouragement. His patience and tact have inspired me to pursue research with humility. His technical competence and willingness to share it with others have been integral to my success. He is truly is a colleague like no other.

I would also like to thank my fellow lab mates in the Cyber Forensics Innovation Lab. All my research is the culmination of many conversations, debates, and laughs that we shared. I especially want to thank Mingxuan Yao and Ranjita Pai Sridhar. You both have directly impacted every positive aspect of my research. Your willingness to be critical of my work and propose new and exciting approaches to problem-solving has been instrumental in my success. More importantly, I am thankful for your friendship and time spent with my family and me outside of the lab.

I would also like to thank Karlin and Marion Fuller (my parents) and Todd and Sheree Overby (my in-laws) for all their prayers and support. I appreciate the conversations, the interest, the care, and love that you have shown. I am so blessed to have you in my life.

Lastly, I dedicate this dissertation to my Ladybug, my bride, Leigh. Words do not provide adequate means of communicating my thankfulness for your partnership. Your love, encouragement, and commitment to our precious daughters and me are priceless, and I am forever thankful. I am blessed to be your husband, and I love you. To my sweet Gloria and Esther, you have brought so much joy and perspective into my life. I am so thankful for two wonderful and precious daughters. I love you both very much.

## TABLE OF CONTENTS

|  |     |
|--|-----|
| <b>Acknowledgments</b> . . . . .   | v   |
| <b>List of Tables</b> . . . . .  | x   |
| <b>List of Figures</b> . . . . .   | xii |
| <b>Chapter 1: Introduction</b> . . . . .   | 1   |
| 1.1 Motivation . . . . .   | 1   |
| 1.2 Thesis Statement . . . . .   | 2   |
| 1.3 Research Scope and Outline . . . . .   | 2   |
| 1.3.1 C3PO: Large-Scale Study of Covert Monitoring of Command &<br>Control Servers via Over-Permissioned Protocol Infiltration . . . . . | 2   |
| 1.3.2 R2D2: Is That Malware Reading Twitter? Towards Understanding<br>and Preventing Dead Drop Resolvers on Public Web Apps . . . . .    | 3   |
| <b>Chapter 2: Related Work</b> . . . . .   | 5   |
| 2.1 C&C Infiltration and Monitoring . . . . .  | 5   |
| 2.2 Communication Protocol Identification . . . . .  | 6   |
| 2.3 Backward Slicing . . . . .   | 7   |
| 2.4 Symbolic Execution . . . . .   | 8   |
| 2.5 Malware Capability Analysis . . . . .  | 9   |

|   |   |           |
|---|---|-----------|
| 2.6   | Web Application Abuse . . . . .                   | 10        |
| <br>  |   |           |
| <b>Chapter 3: C3PO: Large-Scale Study of Covert Monitoring of Command &amp; Control Servers via Over-Permissioned Protocol Infiltration . . . . .</b> |   | <b>11</b> |
| 3.1   | A Motivating Example . . . . .                    | 13        |
| 3.2   | Measurement Pipeline . . . . .                    | 18        |
| 3.2.1   | Dynamic Memory Image Extraction . . . . .         | 19        |
| 3.2.2   | Over-Permissioned Bot Identification . . . . .    | 20        |
| 3.2.3   | Infiltration Vector (IV) Identification . . . . . | 23        |
| 3.2.4   | C&C Monitoring Capabilities . . . . .             | 24        |
| 3.3   | Validating our Techniques . . . . .               | 27        |
| 3.3.1   | Protocol Identification Evaluation . . . . .      | 29        |
| 3.3.2   | C&C Monitoring Capabilities Evaluation . . . . .  | 32        |
| 3.4   | Large-scale Deployment . . . . .                  | 32        |
| 3.4.1   | Post Deployment Dataset Highlights . . . . .      | 33        |
| 3.4.2   | Over-Permissioned Bot Landscape . . . . .         | 34        |
| 3.4.3   | C&C Monitoring Capabilities at Scale . . . . .    | 37        |
| 3.4.4   | Ranking Over-Permissioned Bot Families . . . . .  | 40        |
| 3.4.5   | Packed Malware . . . . .                          | 40        |
| 3.5   | C3PO Applied . . . . .                            | 41        |
| 3.5.1   | Ethical Considerations . . . . .                  | 42        |
| 3.5.2   | Case Study 1: Steam . . . . .                     | 42        |
| 3.5.3   | Case Study 2: Detplock . . . . .                  | 43        |
| 3.6   | Discussion and Limitations . . . . .              | 45        |

|  |  |           |
|--|--|-----------|
| 3.6.1  | Domain Generating Algorithms . . . . .                       | 45        |
| 3.6.2  | Subverting Dynamic Memory Image Extraction . . . . .         | 45        |
| 3.6.3  | Custom Low-level Protocol Implementations . . . . .          | 46        |
| 3.7  | Conclusion . . . . .   | 46        |
| <br>   |  |           |
| <b>Chapter 4: R2D2: Is That Malware Reading Twitter? Towards Understanding<br/>and Preventing Dead Drop Resolvers on Public Web Apps . . . . .</b> |  | <b>48</b> |
| 4.1  | Overview . . . . .   | 50        |
| 4.1.1  | Running Example - Razy . . . . .                             | 50        |
| 4.2  | Design . . . . .   | 54        |
| 4.2.1  | Dead Drop Resolver Candidate Identification . . . . .        | 54        |
| 4.2.2  | Dead Drop Resolver Confirmation . . . . .                    | 58        |
| 4.2.3  | Decoder Identification . . . . .                             | 60        |
| 4.3  | Validating Our Techniques . . . . .                          | 65        |
| 4.3.1  | Dead Drop Resolver Identification and Confirmation . . . . . | 67        |
| 4.3.2  | Decoding Algorithm Comparison . . . . .                      | 68        |
| 4.3.3  | Decoding Algorithm Identification . . . . .                  | 73        |
| 4.4  | Dead Drop Resolver Findings . . . . .                        | 74        |
| 4.4.1  | Dead Drop Resolver-Based Malware Discoveries . . . . .       | 75        |
| 4.4.2  | Decoders Identified . . . . .                                | 78        |
| 4.4.3  | Towards Remediation . . . . .                                | 79        |
| 4.4.4  | Packed Dead Drop Resolver-Based Malware . . . . .            | 81        |
| 4.5  | Discussion . . . . .   | 82        |
| 4.5.1  | Adversarial Response . . . . .                               | 82        |



|  |  |           |
|--|--|-----------|
| 4.5.2  | Uncooperative Web App Providers . . . . .                          | 85        |
| 4.5.3  | Domain Generation Algorithm Domain Origin Identification . . . . . | 85        |
| 4.5.4  | A Subtler Case of Implicit Flows . . . . .                         | 85        |
| 4.6  | Conclusion . . . . .   | 86        |
| <b>Chapter 5: Conclusion and Future Work . . . . .</b> |  | <b>87</b> |
| 5.1  | Goals . . . . .  | 87        |
| 5.1.1  | Scalable Malware Analysis . . . . .                                | 88        |
| 5.1.2  | Reusable Malware Logic . . . . .                                   | 88        |
| 5.1.3  | Validating our Approach . . . . .                                  | 89        |
| 5.2  | Challenges . . . . .   | 90        |
| 5.2.1  | Scalable Malware Analysis . . . . .                                | 90        |
| 5.2.2  | Reusable Malware Logic . . . . .                                   | 90        |
| 5.2.3  | Validating our Approach . . . . .                                  | 90        |
| 5.3  | Solutions and Results . . . . .                                    | 91        |
| 5.3.1  | C3PO . . . . .   | 91        |
| 5.3.2  | R2D2 . . . . .   | 92        |
| 5.4  | Future Work . . . . .  | 92        |
| <b>References . . . . .</b>                            |  | <b>93</b> |

## LIST OF TABLES

|      |   |    |
|------|---|----|
| 3.1  | C3PO’s Analysis of the Sanny Malware. . . . .   | 15 |
| 3.2  | Over-Permissioned Protocols . . . . .   | 21 |
| 3.3  | C&C Monitoring Capabilities . . . . .   | 26 |
| 3.4  | Validating Protocol Identification. GT represents the ground truth compared with C3PO’s results to identify the TP, FP, and FN metrics. . . . .                                       | 28 |
| 3.5  | Validating C&C Monitoring Capabilities Identification. GT and C3PO represent the number of manually verified and automated capability identifiers per category, respectively. . . . . | 31 |
| 3.6  | Distribution of Over-Permissioned Bots Identified During the Large-Scale Study. . . . .   | 35 |
| 3.7  | C3PO Identification of C&C Monitoring Capabilities Mapped to Over-Permissioned Protocols. . . . .   | 38 |
| 3.8  | Evolution of the Top 10 Families of Over-Permissioned Bots Detected in our Dataset. . . . .   | 39 |
| 3.9  | Packers Encountered in our C3PO’s Dataset. . . . .  | 41 |
| 3.10 | C3PO’s Steam Malware Analysis Results. . . . .  | 42 |
| 3.11 | C3PO’s Detplock Malware Analysis Results. . . . .   | 44 |
| 4.1  | Defensive Evasion APIs Considered in R2D2. . . . .  | 56 |
| 4.2  | Common Malware Decoding Algorithms. . . . .   | 62 |
| 4.3  | Validating Dead Drop Domain Candidate Identification and Confirmation. . . . .  | 66 |

|      |  |    |
|------|--|----|
| 4.4  | Baseline Comparison for Decoding Algorithm Similarity via Symbolic Expressions Matching. . . . . | 69 |
| 4.5  | Baseline Comparison for Decoding Algorithm C/C++ Source Code via Moss. . . . .                   | 70 |
| 4.6  | Baseline Comparison for Decoding Algorithm Similarity via Symbolic Expressions. . . . .          | 71 |
| 4.7  | Validating Decoder Identification. . . . .   | 74 |
| 4.8  | Distribution of Web App Domains used for DDR-based Malware Across our Dataset. . . . .           | 76 |
| 4.9  | The Number of Occurrences of Decoders in the DDR-based Malware. . . . .                          | 78 |
| 4.10 | Packed Dead Drop Resolver-Based Malware. . . . .   | 81 |
| 4.11 | Bitcoin Wallet IDs (1/3). . . . .  | 82 |
| 4.12 | Bitcoin Wallet IDs (2/3). . . . .  | 83 |
| 4.13 | Bitcoin Wallet IDs (3/3). . . . .  | 84 |

## LIST OF FIGURES

|     |  |    |
|-----|--|----|
| 3.1 | C3PO-enabled Covert Monitoring of Sanny. . . . .   | 15 |
| 3.2 | C3PO Measurement Pipeline: Dynamic Memory Image Extraction: Executes the malware under instrumentation and captures memory images; Bloated Bot Identification: Identifies protocol invocation points resulting in call sites for all; Infiltration Vector Identification: Uses Iterative Selective Symbolic Execution to extract infiltration vectors used to spoof communication; Flippable Capability Identification: Uses API-based capability modeling to reveal the composition and contents of the C&C infrastructure; Covert Monitoring: Post-infiltration analysis guided by flippable malware capabilities. . . . . | 17 |
| 3.3 | C3PO’s Infiltration Vector Identification of Sanny. . . . .  | 25 |
| 3.4 | Number of Over-Permissioned Protocols Per Bot From 2006-2020. . . . .  | 33 |
| 4.1 | Twitter Message Retrieved by Razy. . . . .   | 52 |
| 4.2 | R2D2 Measurement Pipeline: Taking a malware binary, C3PO uses <i>DDR Candidate Identification</i> via concolic exploration to reveal connected web apps. C3PO then uses concolic taint propagation toward <i>DDR Confirmation</i> . After confirming DDR capability, C3PO conducts <i>Decoder Identification</i> by concretely localizing decoders in the malware before confirming via symbolic expression matching. . . . .  | 53 |
| 4.3 | DDR Malware Trends Since 2017. . . . .   | 75 |
| 4.4 | A Pastebin Account Removed. . . . .  | 79 |
| 4.5 | Response From WordPress. . . . .   | 80 |
| 4.6 | Response From Twitter. . . . .   | 81 |

## SUMMARY

Automated solutions for identifying Command and Control (C&C) domain resolution techniques and leveraging them for botnet monitoring are not scalable and error prone. Thus, malware proliferates, and botnets continue to damage victim systems globally. This dissertation posits that authorities can leverage reusable malware binary logic to enable automated and scalable opportunities for botnet counteraction. This dissertation presents C3PO and R2D2 measurement pipelines that identify reusable malware logic and studies the (1) evolution of over-permissioned protocols in 200k malware spanning 15 years and (2) under-explored DDR technique in 100k malware spanning five years. C3PO identified 62,202 over-permissioned bots across 8,512 families identifying infiltration vectors that allow C3PO to spoof bot-to-C&C communication. C3PO also identified 443,905 C&C monitoring capabilities, which reveal the composition and contents of the C&C server to guide monitoring post infiltration. We deployed C3PO on two bots with live C&C servers validating its ability to identify over-permissioned protocols, infiltrate C&C servers, and leverage C&C monitoring capabilities to achieve covert monitoring. C3PO also identified over 2500 files containing victim information, additional malicious payloads, exploitation scripts, and stolen credentials, providing legally admissible evidence to engender counteraction attempts. Armed with C3PO, authorities can pursue disruptions and takedowns of over-permissioned protocol-based botnets. Next, R2D2 targets the disruption and take-down of DDR-based botnets. During its analysis of 100k malware, R2D2 revealed 10,170 DDR malware from 154 families. R2D2 also showed the type of encoding used, providing authorities with rapid means to decode C&C server domains, with String Parsing and Base64 being the most common. I reported all our findings to web app providers, and they confirmed them and took action against the 9,155 DDRs (90% of DDR malware discovered).

# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation

Counteracting malicious cyber actors is a lop-sided battle since authorities, incident responders, or network defenders must always be accurate at thwarting attacks. At the same time, the malicious actors only need to be right once to establish botnets successfully. Responding to this highly asymmetric situation is grounded in how much defenders know about the malicious actor's next steps and how quickly they can act on it. Thus, maximizing the time advantage of this information has driven academic institutions, major corporations, and government agencies to pursue and employ techniques to accurately monitor botnets to glean enough information to support and enable a rapid and robust response.

Command and Control (C&C) server monitoring is a fundamental enabler of botnet disruption and takedown occurring before any action is taken and after to gauge the success of counteraction attempts. The first step in monitoring C&C servers is to locate their domain, which their bots resolve statically or dynamically. Unfortunately, automatic solutions for identifying C&C domain resolution techniques and leveraging them for botnet monitoring are not scalable and error prone. Thus, malware proliferates, and botnets continue to damage victim systems globally.

Before the work in this dissertation, researchers and authorities alike were forced into cross-domain investigations of numerous malware variants using static and dynamic analysis, network trace analysis, and Internet-wide scans to profile botnet activity pre-counteraction, then repeating similar steps to ensure the efficacy of their approach [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. However, these early approaches suffered from inaccuracy due to their coarse-grained techniques [11, 12] or easy detection because of their *nosiness* [13, 14, 15, 16, 17, 2, 4, 5, 6,

7, 18], prompting defensive evasion by C&C orchestrators [12, 19, 20]. An ideal solution should be accurate, stealthy, and submit to single domain analysis towards scalability.

## 1.2 Thesis Statement

This dissertation posits that *authorities can leverage reusable malware binary logic to enable automated and scalable opportunities for botnet counteraction*. To this end, we developed C3PO [21] and R2D2 [22] which enable botnet disruption and takedown through C&C infiltration, monitoring, and in collaboration with website application service providers, the removal of publicly accessible but hidden dynamically-resolved C&C server domain names to dismantle botnets.

## 1.3 Research Scope and Outline

This dissertation is divided into two main research thrusts targeting two main malware categories. This first thrust, C3PO, targets malware that statically resolves their C&C server domain names or Internet Protocol (IP) addresses. More importantly, these malware also rely on standardized communication protocols that adhere to predefined specifications (i.e., entire communications protocols are *baked* into the malware). Conversely, the second thrust presents R2D2, which targets malware that dynamically resolves the C&C server domain names or IPs. However, our investigation found that authorities have heavily counteracted the three state-of-the-art approaches in dynamic resolutions techniques, so C&C orchestrators are adopting web applications in malware to more robustly resolve C&C server addresses. We provide a more in-depth summary of each research thrust below.

### 1.3.1 C3PO: Large-Scale Study of Covert Monitoring of Command & Control Servers via Over-Permissioned Protocol Infiltration

Current techniques to monitor botnets towards disruption or takedown are likely to result in inaccurate data gathered about the botnet or be detected by C&C orchestrators. Seeking

a covert and scalable solution, we look to an evolving pattern in modern malware that integrates standardized *over-permissioned* protocols, meaning that they provide feature-rich and unfettered access to the server beyond the subset of features implemented by a given client. These over-permissioned protocols expose privileged access to C&C servers. We implement techniques to detect and exploit these protocols from over-permissioned bots toward covert C&C server monitoring. Our empirical study of 200,000 malware captured since 2006 revealed 62,202 over-permissioned bots (nearly 1 in 3) and 443,905 C&C Monitoring Capabilities, with a steady increase of over-permissioned protocol use over the last 15 years. Due to their ubiquity, we conclude that even though over-permissioned protocols allow for C&C server infiltration, the efficiency and ease of use continue to make them prevalent in the malware operational landscape. Chapter 3 presents C3PO, a pipeline that enables our study and empowers incident responders to automatically identify over-permissioned protocols, infiltration vectors to spoof bot-to-C&C communication, and C&C Monitoring Capabilities that guide covert monitoring post infiltration. Our findings suggest the over-permissioned protocol weakness provides a scalable approach to covertly monitor C&C servers, which is a fundamental enabler of botnet disruptions and takedowns.

### 1.3.2 R2D2: Is That Malware Reading Twitter? Towards Understanding and Preventing Dead Drop Resolvers on Public Web Apps

Authorities are increasingly countering DGA botnets, prompting malware authors to shift to Dead Drop Resolvers (DDRs) to hide C&C server domain names. Unlike DGAs, DDRs allow malware to migrate to unpredictable C&C rendezvous points (domain names or IP addresses) by simply posting on public web applications. Hiding in plain sight, malware authors encode these publicly accessible C&C rendezvous points so authorities remain unaware of their true intent. Now, authorities must undergo extensive analysis to decode the C&C domain before taking action. Chapter 4 aims to study this DDR adoption trend. We developed R2D2, an automated DDR malware analysis pipeline, by analyzing 100k



malware. R2D2 identified 10,170 DDR malware from 154 families and revealed the DDR encoding schemes used, providing authorities with a rapid means to decode C&C server domains. I reported our findings to web app providers, who confirmed and took action against 9,155 DDRs (90% of DDR malware discovered). Of the remainder, web app providers previously took down the accounts for 774 malware, and we are awaiting a response for the final 241 DDRs.

The rest of this dissertation is organized as follows. The related works are discussed in Chapter 2. Chapter 3 presents C3PO, an automated malware analysis framework to study the use of over-permissioned protocols in malware. Chapter 4 will discuss the design and implementation of R2D2 to study Dead Drop Resolvers and how C&C orchestrators use them to hide their C&C server rendezvous points for malware to resolve dynamically. Lastly, we conclude this dissertation and discuss future work in Chapter 5.

## **CHAPTER 2**

### **RELATED WORK**

Previous work related to this research can be divided into six broad categories: C&C Infiltration and Monitoring (Section 2.1), Communication Protocol Identification (Section 2.2), Backward Slicing (Section 2.3), Symbolic Execution (Section 2.4), Malware Capability Analysis (Section 2.5), and Web Application Abuse (Section 2.6).

#### **2.1 C&C Infiltration and Monitoring**

C&C monitoring research has mainly focused on P2P botnets. Specifically, Andreisse et al. [12] investigates the myriad of ways botnet monitoring occurs and potential defenses that the next generation of botnets may employ. Echoing this finding, Böck et al. [23] noted that monitoring countermeasures will continue to progress, making any attempts to successfully monitor botnets infeasible. In fact, Karuppayah et al. [24] stated these challenges in monitoring P2P botnets and developed two mechanisms to accurately detect botnet monitors solidifying the claims of present difficulties in botnet monitoring. In response, Karuppayah et al. [25] later introduces a less invasive monitoring technique to crawl P2P botnets, significantly outperforming other crawling methods effectively.

Although the focus has primarily been directed towards P2P botnets, other works have investigated individual malware of families towards infiltration. Dispatcher [26] analyzes botnet protocols to enable infiltration. The authors rewrite the C&C messages for active botnet infiltration to do this. Specifically, Dispatcher extracts the message format and semantics for messages sent and received to extract protocol information. In conjunction with "Google hacking" for information gathering, these techniques are applied to the MegaD botnet. While successful, Dispatcher does not consider the plethora of commodity protocols used in malware.

Lastly, domain seizure approaches provide another relevant avenue to monitoring by taking over the botnet [2]. Stone-Gross et al. [18] took control of the Torpig botnet for ten days after reverse-engineering the domain generating algorithm used in the malware. Similar techniques have been proposed [5, 6, 7], with temporary successes against targeted botnets. However, the infiltrations are short-lived and require detailed reverse engineering efforts to understand domain generating algorithms to predict future candidate domains for seizure. Motivated by the temporariness of these approaches, Nadji et al. [4] proposed a takedown analysis and recommendation system to provide the means to analyze previous botnet disruption and takedown attempts which influences the development of a takedown recommendation engine. This engine automatically enumerates the botnet's C&C infrastructure and suggests appropriate actions to make infiltration and disruption attempts more feasible and practical.

## **2.2 Communication Protocol Identification**

Several works infer protocol formats based on network traces [27, 28, 29]. Notably, Ma et al. [27] classifies network flows according to the application-layer protocols they use. The authors opt to use flow content instead of the traditional flow-external features (e.g., packet sizes, header fields, etc.) because it provides distinguishable information about the specific application-layer protocols being used, even when faced with tunneling or dynamic port allocation. Similarly, Cui et al. [28] relies on network traces to enable protocol reverse engineering. Yet, this approach uses the flow-external features to identify message formats of an application by inferring idioms that are standard in many application-layer protocols. Next, Dreger et al. [29] recognized the shortcomings of protocol identifications using port numbers noting that much of today's network traffic is not classifiable in this way. Thus, the authors designed and implemented a network intrusion detection system extension that dynamically analyzes application-layer protocols before deploying additional analyzers for fine-grained analyses. While these approaches can be effective, sole reliance on network

traces can be problematic for protocol inference if captured network traces are incomplete or non-existent.

Different from the above approaches, but with the common goal of protocol inference, other works infer protocols based on their understanding of how the binary processes network messages [30, 31, 26, 32]. For instance, Caballero et al. [30] extracts protocol information from the binary using *shadowing*. This technique monitors the execution of the binary as it processes input which reveals the type(s) of protocol(s) being used. Similarly, Lin et al. [31] monitors program execution. The authors developed AutoFormat to obtain the execution context of a running program and use it to cluster protocol fields and their relations. Lastly, Prospex [32] automatically infers stateful protocol specifications using protocol state machines revealing the order of message delivery, providing insights into the underlying protocol.

### **2.3 Backward Slicing**

Slicing is a widely used technique because it simplifies a program allowing for the analysis of only relevant portions based on user criteria [33]. Slicing has been used for reverse engineering [34], software maintenance [34, 35], software debugging and testing [36, 37], and program comprehension [38, 39]. Of the various types of slices available, backward slicing is uniquely tailored to reveal all dependencies of a specific operation in a program.

Backward slicing reveals all control and data dependencies for a specific operation in the binary [40, 41]. Specifically, a backward slice is a backward traversal of a program dependence graph composed of control dependence and data dependence graphs. However, an inherent challenge with backward slicing is path-insensitivity, where irrelevant dependencies are added to the slice. These irrelevant dependencies cause slices that are too big creating intractability and diluting the overall accuracy. Therefore, several heuristics and techniques have been developed to address these problems [42, 43, 44]. Of note, Jaffar et al. [42] discovered a general occurrence of imprecise slices bigger than expected and not

very useful. As such, the authors recommend path-sensitive slicing driven by symbolic execution. Since symbolic inputs are used to drive exploration, if any branch becomes unsatisfiable, removing the path that follows prunes the slice, resulting in a more accurate and path-sensitive slice. Similarly, Srinivasan and Reps [44] pursued more accurate slices but chose an algorithmic approach. Their approach works at the microcode level, providing more granularity and ultimately improving the precision of the program slices.

## 2.4 Symbolic Execution

Symbolic execution is used to find software bugs [45, 46, 47], generate test cases [48, 49, 50], and improve the execution of dynamic analysis [51, 52, 53]. Symbolic execution has also been applied to side-channel research [54], firmware analysis [55], cryptographic software validation [56], emulator testing [57], and binary patching [58]. Many of these approaches implement the whole-binary approach seeking to expose all paths for traversal, and are often enabled by simple heuristics. Specifically, MalMax [53], X-Force[51], and J-Force [52] enable the analysis of dynamic code by exposing hidden behaviors using forced execution. All three techniques explore branches, including those that are unsatisfiable. While MalMax uses backtracking and reversing to enable path exploration, J-Force mutates satisfiable branch predicates to explore unvisited paths. However, although X-Force explores infeasible paths, it does not explore all because forcing all paths can induce path explosion, a limitation of symbolic execution. Similarly, Smartgen[59] proposed Selective Symbolic Execution to expose URLs of mobile applications. Smartgen first extracts paths constraints of interests that are subsequently solved to selectively explore paths. The results are used as input values to UI elements, allowing for input-dependent selective path exploration. Most recently, [60] reduces path explosion by using the degree of concreteness technique to identify capability-driven paths and does not require an intact malware binary or prior knowledge of a program's input and environment, avoiding restrictive assumptions for symbolic execution.

## 2.5 Malware Capability Analysis

Several works use behavior analysis [61, 62, 63], behavior modeling [64, 65, 66], and network traffic observation [67, 68] to identify malware. Techniques such as behavior graph generation [69] or network observation indicate suspicious activity with varying degrees of confidence [67, 68] but may not identify specific capabilities. Those that detect malware capabilities [63, 69, 65, 64] are either specific to the Android framework or use dynamic analysis to identify just enough capabilities for malware detection.

Martignoni et al [63] break down high-level actions found in a binary into their respective low-level actions. Actions such as "keystroke logging" include various low-level commands. Next, the work done by Kolbitsch et al [69] also uses dynamic analysis and is used for malware detection. The approach used observed sequences of system calls when dynamically executing to form behavior graphs for malware. These behavior graphs represent the entire malware binary, but the individual capabilities of some malware functionalities can still be observed in the graphs. Reanimator [64] acknowledges limitations with dynamic analysis and instead uses a combination of that and statically searches the binary to find hidden functionalities that an analyst is looking for. Reanimator attempts to automatically create functionality-aware models, which can be used to find the capabilities of the malicious program. Aafer et al. [65] analyze Android malware to find differences between commonly used APIs in benign and malicious binaries.

Deng et. al. [67] reveal malware behavior through network traffic observation. They observe scanning, propagating, and downloading capabilities. We also see similar work with Anubis [68], which tracks specific Windows APIs and data flows when recording network traffic. However, the identified capabilities in both of these systems are not granular enough to pinpoint specific types of information being exfiltrated from the infected system.

## 2.6 Web Application Abuse

When web applications were introduced, malicious actors immediately began developing techniques to attack web app users. This early exploitation prompted research into the security of web apps that were focused primarily on protecting legitimate users [70, 71, 72]. More recently, attackers have begun to abuse web apps to perpetrate cybercrimes prompting investigators to also focus on identifying instances of web app abuse and providing solutions toward remediation.

Clark et al. [73] is one of the first to explore web apps used to orchestrate attacks. They discuss the ability to launch attacks from within the web apps to external targets and use two experiments to demonstrate the practicality of denial of service (DoS) attacks. They also discuss the difficulty of using current botnet detection methods on the new web app-based botnets noting the need for new techniques to combat this emerging threat. This early work influenced research that investigated social network abuse. Specifically, Badis et al. [74] specifically investigated the detection of botnets that use web apps for DoS attacks based on system metrics captured. Next, Lingam et al. [75] identified malware behavioral similarities and proposed a model to detect Twitter botnets. Finally, Pantic et al. [76] demonstrated the use of steganography for secret communication through Twitter for botnet command and control.

Instead of malware analysis, other works opted to analyze network traffic to identify abused cloud repositories [77] or cloud app abuse to enable C&C botnets [78, 79, 80]. Most recently, Netskope reported that up to 66% of malware downloads originated from cloud apps [81]. As a result of these works, MITRE ATT&CK [82] suggests traditional intrusion detection systems for either blocking malicious traffic to benign websites, presupposing that authorities can identify malware-specific traffic to benign web apps from all other benign traffic, or restricting all access to web-based content used by malware.

### CHAPTER 3

## C3PO: LARGE-SCALE STUDY OF COVERT MONITORING OF COMMAND & CONTROL SERVERS VIA OVER-PERMISSIONED PROTOCOL INFILTRATION

Botnet disruptions and takedowns are driven by Command and Control (C&C) server monitoring before any action is taken and after to gauge success. This means that disruption or takedown attempts are not only provably necessary, but must be targeted and effective [2, 3, 4, 5, 6, 7, 8, 9, 10]. Modern approaches can be categorized as passive or active monitoring. Passive monitoring (e.g., sensor node injection) is coarse-grained and may not give accurate insights into the botnet [11, 12], i.e., the number and location of the victims and the extent of damages incurred. It also requires a full reverse engineering effort to maintain sensor nodes making this approach not widely used [12]. Therefore, active monitoring is the preferred approach [11, 2], generally providing better insights into botnet operations. However, active monitoring techniques, including remote penetration testing [13, 14, 15, 16, 17] and domain seizure [2, 4, 5, 6, 7, 18], are *noisy* making them easily detectable. Seeking a better solution, this research proposes that standard protocols, which are increasingly used by botnets, can be leveraged for *general and covert* C&C server monitoring.

In previous botnet disruption and takedown attempts, authorities first monitored the C&C server to prove malware as the catalyst for incurred damages before legal permission was granted for counteraction [83]. Yet, accurate monitoring goes beyond determining the legality of counteraction. For example, to protect the 2020 election, Microsoft took down 120 of 128 Trickbot C&C servers [84]. Accurately identifying C&C servers pre-takedown (profiling), then tracking successes post takedown (validation), required an in-depth understanding of the peers in the botnet, C&C server locations, and weaknesses to leverage for botnet disruption. Therefore, successful monitoring must result in accurate, legally-



admissible information gathered during profiling and remain covert to avoid discovery by C&C orchestrators, prompting defensive evasion or hardening [12, 19, 20]. An ideal solution should provide authorities with a means to access the C&C server under the guise of normal bot operation.

As the *end-host agents* of a C&C orchestrator, bots are entrusted with C&C server access. In fact, attackers are entirely dependent on the information exfiltrated by bots to gain situational awareness in a victim’s network. To enable command and control, bots use standard protocols for file transfer, data storage, and message-based communication. However, many standard protocols are over-permissioned, meaning that they provide feature-rich and unfettered access to the server beyond the subset of features implemented by a given client. A similar trend has been observed in benign software where over-permissioned client-side protocols lead to unauthorized server access [85, 86, 87, 88]. This prompted our key insight: *over-permissioned protocols combined with the trust C&C servers place in their bots expose a scalable opportunity for covert monitoring of C&C servers through protocol infiltration.*

To explore this insight, a systematic study is needed to identify the evolution of over-permissioned protocol use in malware. Moreover, to conduct such a study, the analysis must be scalable, reproducible, and provide the requisite information to covertly monitor C&C servers through over-permissioned protocol infiltration. The study must expose over-permissioned protocols, how they are being used, and the associated levels of access and recoverable data on the C&C server. Finally, an automated pipeline must be made available to enable the authorities to take action on these common malware weaknesses in future botnet outbreaks.

We turned our attention to how the authorities could recover C&C server access privileges from over-permissioned bots (bots using over-permissioned protocols) allowing them to spoof bot-to-C&C communication. To this end, we designed and implemented C3PO<sup>1</sup>,

---

<sup>1</sup>C3PO: Covert Monitoring of C&C Servers via Protocol InfiltratiOn

an automated memory-image-based symbolic analysis measurement pipeline. C3PO analyzes a malware memory image to identify (1) over-permissioned protocols, (2) infiltration vectors (i.e., authentication information to spoof bot-to-C&C communication), and (3) C&C monitoring capabilities (i.e., capabilities in the end-host bot that reveal the C&C server’s composition and content to guide covert monitoring post infiltration).

Through our collaboration with Netskope, the leading Secure Access Service Edge (SASE) provider, which provides cloud security and networking to more than 30% of the Fortune 100, we used C3PO to study the evolution of over-permissioned protocol use in 200k malware spanning back 15 years. C3PO uncovered 62,202 over-permissioned bots ( $\approx 1$  in 3). Our empirical measurement revealed several interesting findings: FTP is the most prevalent over-permissioned protocol found in over 79% of all over-permissioned bots. C3PO also identified 443,905 C&C monitoring capabilities (an average of 7 per bot), enabling victim profiling, evidence collection from spyware, and even client-side code reflection. This trend has only increased since 2006, with over 8,000 over-permissioned bots appearing per year in 2018 and 2019. Furthermore, recent bots (since 2015) implemented as many as 3 over-permissioned protocols.

Finally, we present two case studies to demonstrate covert C&C server monitoring through protocol infiltration. We were careful to follow ethical guidelines and adhere to applicable laws when conducting this study. Covert monitoring succeeded and revealed the number of files, their contents, and validation of information inferred by the C&C monitoring capabilities, which will support future botnet disruption and takedown attempts. We are working with Netskope towards the disclosure and remediation of the identified C&C servers.

### **3.1 A Motivating Example**

Botnet disruptions and takedowns rely on accurate C&C server monitoring to profile the botnet beforehand and validate successes after. Consider Sanny, an APT that targets gov-

ernment agencies through spearfishing. After infection, Sanny hijacks Windows service components to enable persistence, deletes dropped files to cover their tracks, and conducts sensitive data exfiltration. The Sanny botnet survived takedown attempts in 2013 [89] and persists today. After botnet monitoring began to fail, an extensive investigation was conducted in 2018, revealing Sanny's C&C server update [90], but this required a tedious manual analysis.

The authorities reverse engineered dropped malicious files to investigate the new Sanny variant. At the time, authorities found never-before-seen FTP APIs and authentication credentials throughout the malware binary and configuration files on the infected system, revealing the update to the Sanny C&C server. However, since no further action was taken, they likely did not realize the leverage this provided for covert C&C server infiltration. If they did, the authorities could have also identified the malware capabilities that rely on FTP for interaction with the C&C server. This would have allowed them to reinstate monitoring of the botnet's spread by extracting victim profiles and new bot command updates, all under the covert guise of a trusted FTP connect.

Armed with our key insight, C3PO monitors the C&C server by first identifying over-permissioned protocols, FTP in this case, through their invocation points in the malware. Figure 3.1 illustrates the sequence of events toward covert C&C server monitoring. During malware analysis ❶, C3PO identified FTP APIs (e.g., `FTPputFile`) in Sanny which confirmed the updated Sanny C&C server (Table 3.1, Row 1). C3PO then used Iterative Selective Symbolic Execution (iSSE) to extract infiltration vectors (IVs) from FTP APIs ❷, allowing C3PO to spoof bot-to-C&C communication for infiltration while masquerading as a trusted bot (Table 3.1, Row 2).

Had authorities realized the leverage FTP provided for botnet infiltration, they could have monitored victim profiles and new bot command updates. C3PO automatically provides this by identifying C&C monitoring capabilities ❸ revealing the C&C's composition and content that authorities can expect post infiltration. C3PO only targets those capa-

Figure 3.1: C3PO-enabled Covert Monitoring of Sanny.

bilities that are exploitable, i.e., they interact with the C&C server in a way that can be observed by C3PO when it connects to the C&C server using the same protocol. For example, Sanny performs victim profiling by exfiltrating victim locale information, files, and passwords (from Firefox and Microsoft Outlook) via FTP and used code reflection to execute arbitrary commands on the victim system from a file on the C&C server (Table 3.1, Row 3). C3PO maps these capabilities to specific files and directories to monitor on the C&C server via FTP protocol infiltration.

After C3PO extracts the IVs ② and capabilities ③, it actively monitors the C&C server. C3PO can use the IVs (Table 3.1, Row 2) to infiltrate ④ the Sanny C&C server, via the trusted bot-to-C&C channel, and directly locate data from victims ⑤ in the form of files containing infected system information and passwords resulting in peer disclosure (Table 3.1, Row 4) which serves as evidence of computer fraud and abuse. Furthermore, C3PO identified code reflection where the bot orchestrators issue the `chip` command to the

Table 3.1: C3PO’s Analysis of the Sanny Malware.

|  |  |
|--|--|
| <b>Protocol</b>                        | FTP  |
| <b>Infiltration Vectors</b>            | <b>Username:</b> cnix_21072852<br><b>Password:</b> vlasimir2017<br><b>Server:</b> ftp.capnix.com   |
| <b>C&amp;C Monitoring Capabilities</b> | Victim Profiling, File Exfiltration, Password Stealing, and Code Reflection  |
| <b>Covert Monitoring Outputs</b>       | (1)Peer disclosure as victim information is listed as " <code>&lt;Victim ID&gt;_(#report)   UserName   TimeStamp</code> "<br>(2) Code Reflection to update the C&C host name |

bot to trigger the FTP hostname to update. The ability to monitor this transaction ensures that we maintain persistent covert monitoring irrespective of migrating servers.

In contrast to previous works, C3PO gives the ability to identify, assess, and pursue counteraction via scalable covert monitoring. Notably, C3PO does not attempt to find exploitable vulnerabilities in protocol implementations but instead, leverages the inherent capabilities of the protocol.

Figure 3.2: C3PO Measurement Pipeline: Dynamic Memory Image Extraction: Executes the malware under instrumentation and captures memory images; Bloated Bot Identification: Identifies protocol invocation points resulting in call sites for all; Infiltration Vector Identification: Uses Iterative Selective Symbolic Execution to extract infiltration vectors used to spoof communication; Flippable Capability Identification: Uses API-based capability modeling to reveal the composition and contents of the C&C infrastructure; Covert Monitoring: Post-infiltration analysis guided by flippable malware capabilities.

## 3.2 Measurement Pipeline

In collaboration with Netskope, we designed C3PO to study the adoption of over-permissioned protocols in bots and how their use has evolved from April 2006 to June 2020. Our dataset included 200k malware with collection dates spanning back 15 years. This allows us to retroactively deploy C3PO by analyzing each malware sample and give C3PO the vantage point to observe existing trends in the progression of malware development. C3PO identified 62,202 of these as over-permissioned bots totaling 65,739 over-permissioned protocol uses detected across 8,512 malware families. Furthermore, C3PO identified that each bot contains on average 7 C&C monitoring capabilities, totaling 443,905 capabilities identified across our dataset. We hope C3PO provides an automated measurement pipeline to study the over-permissioned bot landscape in the wild and this opportunity for covert botnet monitoring.

Figure 3.2 shows the four phases of C3PO’s automated measurement pipeline that employs a memory-image-based symbolic analysis. Taking a malware binary as input, C3PO conducts *Dynamic Memory Image Extraction* (subsection 3.2.1) by executing the malware under instrumentation and capturing memory images during this execution for analysis. This provides the best vantage point to bypass malware packing and obfuscation. C3PO transitions to static analysis for *Over-Permissioned Bot Identification* (subsection 3.2.2) by identifying invocation points for protocol APIs and protocol keywords/commands (tokens). Next, C3PO uses Iterative Selective Symbolic Execution (iSSE) for *Infiltration Vector Identification* to allow the authorities to spoof bot-to-C&C communication for infiltration (subsection 3.2.3). C3PO then conducts *C&C Monitoring Capabilities Identification* to reveal the composition and content that authorities can expect from the C&C server during infiltration (subsection 3.2.4). Finally, infiltration vectors can be used for *Covert Monitoring* of the C&C servers to pinpoint data inferred by C&C monitoring capabilities enabling botnet monitoring.

### 3.2.1 Dynamic Memory Image Extraction

Malware often employs sophisticated packing and obfuscation techniques that constrain analysis and also inhibit large-scale measurements [91, 92]. Although there are numerous unpacking tools available, modern packing techniques employ robust anti-analysis methods rendering existing solutions mute [91]. While sandboxes or software emulation are viable approaches, they require careful configuration per malware sample/family which is likely to prevent scaling to analyze a large dataset and may accidentally result in introduced errors through incomplete configurations. As a pipeline designed for large-scale measurement, C3PO aims to provide a scalable means of malware analysis through dynamic unpacking and memory image extraction, i.e., taking a snapshot of the malware during normal execution. Ideally, creating a memory image during dynamic execution allows the malware to unpack and deobfuscate itself, leaving C3PO with unpacked and deobfuscated code and execution data to analyze.

Inspired by prior works [91, 60], C3PO extracts multiple memory images during the malware execution by *hooking* Internet and Network (I/N) APIs<sup>2</sup>. This technique is based on two observations: (1) Irrespective of the packing scheme, after unpacking, the malware must invoke I/N APIs to interact with its C&C server. (2) Since recent research has shown that most modern packers have at least two layers of packing [91], if malware memory image extraction is untimely, or at the wrong layer, it will still be packed. Therefore, C3PO extracts multiple memory images by hooking all I/N APIs, as their DLLs are loaded, using a trampoline to replace instructions in the hooked API with a call to our custom code that writes the memory image to a file and returns to the trampoline. Each memory image contains the execution context (i.e., register values, stack, program counter, etc., at the time of memory image extraction) which ensures that malware analysis begins from a valid execution point in the malware.

After extracting malware memory images, C3PO proceeds to the memory-image-based

---

<sup>2</sup>I/N APIs allow the malware to interact with FTP and HTTP protocols to access Internet resources.



analysis to measure the prevalence of over-permissioned protocol use and the leverage they provide to covertly monitor C&C servers.

### 3.2.2 Over-Permissioned Bot Identification

Over-Permissioned bots use over-permissioned protocols that authorities can leverage to covertly monitor C&C servers. We construct a *protocol database* that C3PO can reference as it confirms the invocation of protocol identifiers (APIs and protocol keywords or commands, i.e., tokens) validating protocol use. If the bot is over-permissioned, C3PO outputs the protocol APIs, tokens, and call sites for later analysis.

#### ***Protocol Implementations***

Protocols are implemented using low-level functions or high-level, built-in library functions to achieve the same overall functionality. We, therefore, categorize protocol implementations as low-level ( $L_L$ ) or high-level ( $H_L$ ) for our measurement study.

**$H_L$  Implementations.** Protocol-specific APIs are used for  $H_L$  protocol implementations (e.g., `SQLConnect`), which reduce flexibility in modifying or adding to the protocol but make communications easy and efficient given the built-in APIs.

**$L_L$  Implementations.** Malware authors often hide the use of well-known protocols and prevent an investigator's immediate understanding of the C&C communication routines.  $L_L$  implementations use raw-socket (non protocol specific) APIs (e.g., `send`) in conjunction with official protocol tokens (e.g., `NICK` for the IRC protocol).

Notably, all protocols have  $L_L$  implementations, but only some also have a  $H_L$  implementation. Although custom protocol implementations are feasible, their uniqueness supports signature development making them easier to filter with firewall rules. Thus, C3PO identifies  $H_L$  and  $L_L$  implementations, and could be easily extended to other protocols when deemed necessary for an investigation.

Table 3.2: Over-Permissioned Protocols

| Category                    | Over-Permissioned Protocol                      | Implementation(s) |
|-----------------------------|---|-------------------|
| File Transfer               | File Transfer Protocol (FTP/TFTP)               | $L_L, H_L$        |
|                             | Web Distributed Authoring & Versioning (WebDAV) | $L_L, H_L$        |
|                             | BitTorrent/Micro Transport Protocol ( $\mu$ TP) | $L_L, H_L$        |
| Data Storage                | Mongo Database                                  | $L_L, H_L$        |
|                             | MySQL   | $L_L, H_L$        |
|                             | PostgreSQL                                      | $L_L, H_L$        |
|                             | Object DB Connectivity (ODBC)                   | $L_L, H_L$        |
| Message-based Communication | Internet Relay Chat (IRC)                       | $L_L$             |
|                             | Message Queuing Telemetry Transport (MQTT)      | $L_L, H_L$        |

### ***Protocol Database***

Standard protocols are often used for: (1) file transfer, (2) data storage, and (3) message-based communication. However, their ubiquitous integration into benign software has prompted research into inherent vulnerabilities which has led to unauthorized server access [85, 86, 87, 88]. Noticing a similar trend in malware, we select common over-permissioned protocols discovered in preliminary research, reports from industry experts [93, 82], and related work [94] for our study, as shown in Table 3.2.

Based on the protocols, we constructed a database of all protocol identifiers for C3PO to reference during protocol identification (subsubsection 3.2.2). To construct this database, we developed a web-crawler and targeted it to the respective protocol documentation [95, 96, 97, 98, 99] or manually extracted protocol details to populate the database. However, as other over-permissioned protocols become widely adopted by malware, they can be easily integrated by adding their identifiers to the protocol database. Based on the protocol implementations and the database as a reference, C3PO conducts protocol identification to pinpoint protocol use.

## ***Protocol Identification***

To establish the execution context for malware analysis, C3PO parses the memory images and extracts code pages enabling import address and export directory tables (IAT and EDT) reconstruction. For each memory image, C3PO identifies the code regions to construct a CFG starting at the point the memory image was taken to all reachable code. This results in one CFG per memory image, rooted at the instruction pointer from the memory image. C3PO then creates a Combined CFG (C2FG) by matching overlapping blocks in all CFGs, ensuring no duplication. It then traverses this C2FG to identify all function call sites and compares it against the reconstructed IAT and EDT for a matching API. Although a common challenge in static analysis is resolving indirect function calls, the initial dynamic execution to generate memory images populates concrete function pointers in memory before image extraction, which aids in indirect call resolution. A data dependence graph, built from the C2FG, also resolves additional indirect calls.

**$H_L$  Identification.** To identify  $H_L$  implementations, C3PO traverses the C2FG and resolves call targets. If it encounters an API that is in the protocol database, C3PO stores the call site and the called API. From our example in section 3.1, C3PO detected `FTPPutFile` in Sanny, classifying it as a over-permissioned bot because it uses FTP.

**$L_L$  Identification.**  $L_L$  implementations use raw-socket APIs with a protocol token. When C3PO traverses the C2FG and encounters a call to a raw-socket API, it extracts API arguments to deduce tokens (as described in subsection 3.2.3). If the token is in the protocol database, C3PO stores the call site and the called API/token combination.

C3PO identified 62,202 over-permissioned bots ( $\approx 30\%$ ) in 200k malware. After protocol identification, C3PO continues the analysis to identify information that can be used to spoof bot-to-C&C communication toward infiltration.

### 3.2.3 Infiltration Vector (IV) Identification

Infiltration vectors (IVs) are the credentials used by the bot to connect to the C&C server. To spoof bot-to-C&C communication, C3PO identifies IVs using a combination of *backward slicing* and *iterative selective symbolic execution*.

#### ***Backward Slicing***

C3PO uses the previously identified APIs, call sites, and tokens to first locate the authentication APIs (e.g., `SQLConnect` for  $H_L$  or `send` and a protocol token for  $L_L$ ). C3PO performs backward slicing (of the C2FG) from these API arguments to identify a path to them through the malware. A challenge faced during backward slicing is that API arguments only point to the first byte of the data buffer (e.g., `lpzPassword` for `InternetConnect`) resulting in an incomplete slice. To address this, C3PO generates target instructions by identifying all instructions that were last to write to all bytes of the data buffer.

#### ***Iterative Selective Symbolic Execution (iSSE)***

C3PO symbolically executes along each of the backward slices to the authentication API. Since C3PO is constrained by the slice, symbolic execution is *selective* precluding path explosion while maintaining accuracy. When iSSE reaches the authentication API, it halts to extract API arguments by dereferencing data buffer pointers. If the arguments are concrete, they are decoded to strings and iSSE analysis ends, as the IVs have been found. If they are symbolic, it means the API arguments were passed as parameters from the preceding (calling) function. C3PO, guided by the path, *incrementally* expands the exploration region by starting in the preceding function before re-initiating iSSE. This iterative process continues until the IVs are found. We discuss instances where concretization is not possible in section 3.6. Although execution can begin at the entry point, C3PO is more likely to encounter symbolic loops which can cause resource exhaustion if specific functions in the

malware are computationally complex. Therefore, C3PO starts small (within the function), then incrementally expands to increase the likelihood of argument extraction. Loop handling is still necessary and C3PO employs a loop limiter to exit symbolic loops. However, loop avoidance is still preferred.

Figure 3.3 illustrates C3PO's IV Identification steps for the Sanny malware. C3PO performs backward slicing from the authentication API `InternetConnect`. For each of the authentication API arguments (e.g., `lpzServerName`, `nServerPort`, `lpzUserName`, `lpzPassword`, etc.), C3PO calculates the memory addresses for all bytes of the data buffer using a shadow memory that was populated during data dependency graph generation, a prerequisite for backward slicing. C3PO finds each instruction that was the last to write to each byte of the data buffer (*Target Identification* in Figure 3.3). Using these target instructions, C3PO conducts a backward slice to identify all influencing operations of the data buffer (the blue line through four of Sanny's functions in Figure 3.3). C3PO now traverses each slice using *iSSE* (*iSSE* in Figure 3.3) to extract IVs for all arguments. For Sanny, C3PO extracted the server hostname, username, and password (e.g., Table 3.1, Row 2) after covering only 3 of the 4 functions in the backward slice (the red *iSSE* line in Figure 3.3). Thus, C3PO can spoof bot-to-C&C communication and masquerade as a trusted bot.

#### 3.2.4 C&C Monitoring Capabilities

Bots execute capabilities on the infected systems, some of which can be leveraged to provide covert monitoring. These C&C monitoring capabilities either (1) exfiltrate victim data or (2) allow bot orchestrators to execute arbitrary commands. These capabilities are valuable because the former alerts the authorities about the types and format of data stored on the C&C server, and the latter triggers commands on peer systems for botnet disruption upon infiltration.

To identify these capabilities, C3PO constructs a backward slice from all data exfiltra-

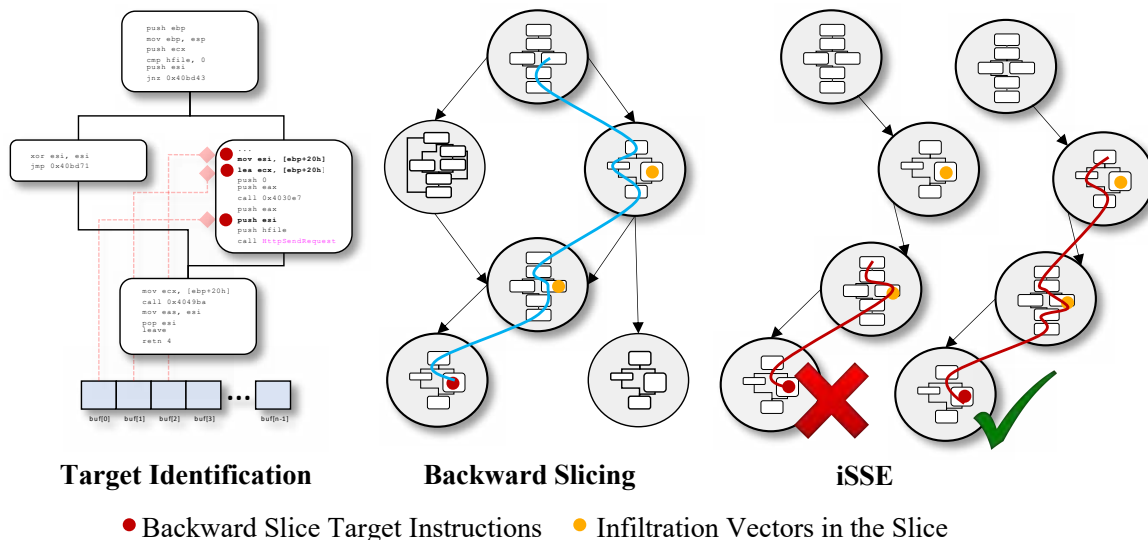


Figure 3.3: C3PO's Infiltration Vector Identification of Sanny.

tion and code reflection targets in the malware. It then performs *API-to-capability mapping* to derive the C&C monitoring capabilities.

### ***Backward Slicing***

C3PO uses the previously identified APIs and call sites to locate data exfiltration (e.g., `HttpSendRequest`) and code reflection (e.g., `ShellExecute`) APIs. With each of these APIs as data sinks, C3PO performs backward slicing. For data exfiltration APIs, it backward slices from the API argument corresponding to the data exfiltration buffer (e.g., `lpOptional` for `HttpSendRequest`). For code reflection APIs, it backward slices from the operation arguments that reveal the C&C command triggers (e.g., `lpOperation` for `ShellExecute`).

### ***API-to-Capability Mapping***

C3PO locates all API calls along each of the backward slices, similar to the technique used in subsection 3.2.2. This gives C3PO API sequences that influence the contents of the data exfiltration buffer or operation argument. These sequences of APIs are then compared against the capability models to identify the C&C monitoring capabilities. The

Table 3.3: C&C Monitoring Capabilities

| Category                  | C&C Monitoring Capabilities   |
|---------------------------|---|
| Browser Password Stealing | (1) Mozilla Stealer<br>Chrome Stealer<br>Internet Explorer Stealer  |
| Service Password Stealing | (2) WiFi Stealer<br>Kerberos Stealer<br>Windows System Stealer  |
| Victim Profiling          | (3) Registry-stored System Details<br>Live System Operating State<br>System OS Details<br>Victim Locale Information |
| Spying, Live Monitoring   | (4) Keylogger<br>Screen Capture<br>Audio Capture  |
| File Exfiltration         | (5) High-level Protocols<br>Raw Socket Transfer   |
| Code Reflection           | (6) Code Reflection   |

capability models are derived by manually reverse engineering known malware and by using the insights from industry reports [82, 100]. In our study, we considered 6 categories of 16 C&C monitoring capabilities, as shown in Table 3.3.

To illustrate, C3PO identifies the victim profiling capability in the Sanny malware (section 3.1). C3PO performs backward slicing from the data sink `HttpSendRequest`. It calculates the memory addresses for all bytes of the sink buffer by referencing the shadow memory that was populated during data dependency graph generation (subsection 3.2.3). C3PO then finds each instruction that was the last to write to each byte of the buffer. Using these target instructions, C3PO conducts a backward slice to identify all influencing operations of the sink buffer. It identifies `GetUserDefaultLCID` and `GetLocaleInfoW` APIs leading up to `HttpSendRequest` API. This API sequence conforms with the capability model for Victim Locale Information, and hence the Sanny malware is classified as having a Victim Profiling Capability.

Note that this capability can be used for covert monitoring because it describes the type of data and format stored on the C&C server which results in immediate victim identification. It also reveals the scope of infection and potential damages incurred (victim credentials provide access to sensitive accounts) providing legally admissible evidence to confirm computer fraud and abuse.

To identify code reflection, the same process holds. However, instead of identifying all APIs along the backward slice, C3PO locates the closest API to the sink that reads incoming information (e.g., `recv`). Once found, C3PO extracts the argument from the buffer to reveal the C&C command that triggered code reflection. This allows the authorities with C&C access to issue the commands to peers in the botnet to trigger arbitrary code execution. This capability goes beyond C&C server monitoring, and instead supports botnet disruption and takedown.

### 3.3 Validating our Techniques

C3PO is implemented in C++ and Python, totaling 11k lines of code leveraging Detours [101] for memory image extraction and angr [102] to support binary analysis with specific applications to protocol identification, backward slicing, and iSSE. We also used the recently released AVClass2 [103], the current state-of-the-art in malware labeling tools, whose predecessor, AVClass [104], has long been relied upon in top-tier research [105, 106, 107, 108].

Before deploying C3PO on the full data set, we validate its accuracy in identifying protocols and leverageable malware capabilities which enable covert and targeted C&C server monitoring. We leave the efficacy of infiltration vector analysis for our case studies (section 3.5) which demonstrate our ability to covertly infiltrate C&C servers. We evaluated C3PO using a ground truth dataset of 35 manually reverse engineered Windows malware from 13 different families, covering all protocols in Table 3.2.



Table 3.4: Validating Protocol Identification. GT represents the ground truth compared with C3PO’s results to identify the TP, FP, and FN metrics.

| Malware<br>(by protocols)            | #Variants | Low-level Identifiers |            |            |           |           | High-level Identifiers |            |            |          |          |
|--------------------------------------|-----------|-----------------------|------------|------------|-----------|-----------|------------------------|------------|------------|----------|----------|
|                                      |           | GT                    | C3PO       | TP         | FP        | FN        | GT                     | C3PO       | TP         | FP       | FN       |
| <b>FTP/TFTP</b>                      |           |                       |            |            |           |           |                        |            |            |          |          |
| Softcnapp                            | 5         | 0                     | 0          | 0          | 0         | 0         | 15                     | 15         | 15         | 0        | 0        |
| Ragebot                              | 2         | 2                     | 2          | 2          | 0         | 0         | 0                      | 0          | 0          | 0        | 0        |
| Blackhole                            | 3         | 3                     | 3          | 3          | 0         | 0         | 0                      | 0          | 0          | 0        | 0        |
| Rbot                                 | 2         | 2                     | 2          | 2          | 0         | 0         | 0                      | 0          | 0          | 0        | 0        |
| <b>Subtotal</b>                      | <b>12</b> | <b>7</b>              | <b>7</b>   | <b>7</b>   | <b>0</b>  | <b>0</b>  | <b>15</b>              | <b>15</b>  | <b>15</b>  | <b>0</b> | <b>0</b> |
| <b>WebDAV</b>                        |           |                       |            |            |           |           |                        |            |            |          |          |
| Equationdrug                         | 2         | 54                    | 42         | 42         | 0         | 12        | 0                      | 0          | 0          | 0        | 0        |
| <b>Subtotal</b>                      | <b>3</b>  | <b>54</b>             | <b>42</b>  | <b>42</b>  | <b>0</b>  | <b>12</b> | <b>0</b>               | <b>0</b>   | <b>0</b>   | <b>0</b> | <b>0</b> |
| <b>BitTorrent/<math>\mu</math>TP</b> |           |                       |            |            |           |           |                        |            |            |          |          |
| Sathurbot                            | 2         | 18                    | 24         | 18         | 6         | 0         | 0                      | 0          | 0          | 0        | 0        |
| Icloader                             | 1         | 0                     | 0          | 0          | 0         | 0         | 21                     | 21         | 21         | 0        | 0        |
| <b>Subtotal</b>                      | <b>3</b>  | <b>18</b>             | <b>24</b>  | <b>18</b>  | <b>6</b>  | <b>0</b>  | <b>21</b>              | <b>21</b>  | <b>21</b>  | <b>0</b> | <b>0</b> |
| <b>MySQL</b>                         |           |                       |            |            |           |           |                        |            |            |          |          |
| Delf                                 | 4         | 0                     | 0          | 0          | 0         | 0         | 24                     | 24         | 24         | 0        | 0        |
| <b>Subtotal</b>                      | <b>4</b>  | <b>0</b>              | <b>0</b>   | <b>0</b>   | <b>0</b>  | <b>0</b>  | <b>24</b>              | <b>24</b>  | <b>24</b>  | <b>0</b> | <b>0</b> |
| <b>MongoDB</b>                       |           |                       |            |            |           |           |                        |            |            |          |          |
| Cstealer                             | 1         | 2                     | 4          | 2          | 2         | 0         | 5                      | 5          | 5          | 0        | 0        |
| <b>Subtotal</b>                      | <b>1</b>  | <b>2</b>              | <b>4</b>   | <b>2</b>   | <b>2</b>  | <b>0</b>  | <b>5</b>               | <b>5</b>   | <b>5</b>   | <b>0</b> | <b>0</b> |
| <b>ODBC</b>                          |           |                       |            |            |           |           |                        |            |            |          |          |
| Zbot                                 | 4         | 0                     | 0          | 0          | 0         | 0         | 60                     | 60         | 60         | 0        | 0        |
| <b>Subtotal</b>                      | <b>4</b>  | <b>0</b>              | <b>0</b>   | <b>0</b>   | <b>0</b>  | <b>0</b>  | <b>60</b>              | <b>60</b>  | <b>60</b>  | <b>0</b> | <b>0</b> |
| <b>PostgreSQL</b>                    |           |                       |            |            |           |           |                        |            |            |          |          |
| Alma                                 | 2         | 0                     | 0          | 0          | 0         | 0         | 5                      | 5          | 5          | 0        | 0        |
| <b>Subtotal</b>                      | <b>2</b>  | <b>0</b>              | <b>0</b>   | <b>0</b>   | <b>0</b>  | <b>0</b>  | <b>5</b>               | <b>5</b>   | <b>5</b>   | <b>0</b> | <b>0</b> |
| <b>IRC</b>                           |           |                       |            |            |           |           |                        |            |            |          |          |
| Softcnapp                            | 5         | 15                    | 15         | 15         | 0         | 0         | 0                      | 0          | 0          | 0        | 0        |
| Ragebot                              | 1         | 6                     | 8          | 6          | 2         | 0         | 0                      | 0          | 0          | 0        | 0        |
| Rbot                                 | 2         | 9                     | 7          | 7          | 0         | 2         | 0                      | 0          | 0          | 0        | 0        |
| Slackbot                             | 4         | 12                    | 12         | 12         | 0         | 0         | 0                      | 0          | 0          | 0        | 0        |
| Delf                                 | 4         | 12                    | 15         | 12         | 3         | 0         | 0                      | 0          | 0          | 0        | 0        |
| <b>Subtotal</b>                      | <b>16</b> | <b>54</b>             | <b>57</b>  | <b>52</b>  | <b>5</b>  | <b>2</b>  | <b>0</b>               | <b>0</b>   | <b>0</b>   | <b>0</b> | <b>0</b> |
| <b>MQTT</b>                          |           |                       |            |            |           |           |                        |            |            |          |          |
| Expiro                               | 3         | 0                     | 0          | 0          | 0         | 0         | 39                     | 39         | 39         | 0        | 0        |
| <b>Subtotal</b>                      | <b>3</b>  | <b>0</b>              | <b>0</b>   | <b>0</b>   | <b>0</b>  | <b>0</b>  | <b>39</b>              | <b>39</b>  | <b>39</b>  | <b>0</b> | <b>0</b> |
| <b>Total</b>                         | <b>35</b> | <b>135</b>            | <b>134</b> | <b>121</b> | <b>13</b> | <b>14</b> | <b>169</b>             | <b>169</b> | <b>169</b> | <b>0</b> | <b>0</b> |

### 3.3.1 Protocol Identification Evaluation

Table 3.4 presents C3PO’s protocol identification evaluation. Columns 1-2 list the malware families (categorized by protocols found in each) and the number of malware variants (*Var*). Columns *Low-level* and *High-level Identifiers* present the ground truth (GT) findings, C3PO’s analysis results of protocol identifiers found, and the true positive (TP), false positive (FP), and false negative (FN) metrics for each, respectively. C3PO correctly (TP) identified 290 (121  $L_L$  +169  $H_L$ ) protocol identifiers. Our GT analysis confirmed 304 (135  $L_L$  + 169  $H_L$ ) of them, revealing 13 FPs, 14 FNs, and an overall accuracy of over 94%.

We then dug into the detection of protocols among all variants. As an example, we identified 4 of the 13 malware families use FTP employing both  $L_L$  and  $H_L$  identifiers. C3PO’s analysis of the Softcnapp, Ragebot, Blackhole, and Rbot malware reported no FTP FPs and FNs.

Upon close inspection, we found that FPs occur when C3PO incorrectly identifies the use of a token (protocol command or keyword). C3PO reported 2 extra IRC tokens in Ragebot due to custom C&C commands which also used the `PASS` keyword (also an IRC command). Similarly, C3PO reported FPs in  $L_L$  implementations of MongoDB (2 false tokens), IRC (5 false tokens), and BitTorrent/uTP (6 false tokens) due to tokens appearing as substrings in other C&C communication. Although adding missed tokens to the protocol database reduces FNs, this is a case-by-case basis. Also, there is a tradeoff between FPs and FNs - allowing and ignoring substrings would increase FPs and FNs, respectively (e.g., some IRC bots use multiple tokens in one message, while in general, FTP bots do not).

Of the 135 manually-verified  $L_L$  identifiers, C3PO produced 14 FNs because of undocumented tokens resulting in 121 of 135 TPs. The only FNs occurred during the IRC and WebDAV protocols identification via their  $L_L$  implementation in the Rbot and Equation-drug samples, respectively. Whenever we encountered undocumented tokens, we subsequently added them to the protocol database. However, we retained our accuracy metrics results pre-modification as it represents a more accurate depiction of C3PO’s protocol iden-

tification capability given the possibility of future undocumented tokens. Overall, C3PO was 94% accurate in identifying protocols making it robust enough to be applied to the large-scale study.

Table 3.5: Validating C&C Monitoring Capabilities Identification. GT and C3PO represent the number of manually verified and automated capability identifiers per category, respectively.

| Malware      | #Variants | Browser Password Stealer |      | Service Password Stealer |      | Victim Profiling |      | File Exfiltration |      | Spying, Live Monitoring |      | Code Reflection |      | Accuracy Metrics |    |    |
|--------------|-----------|--------------------------|------|--------------------------|------|------------------|------|-------------------|------|-------------------------|------|-----------------|------|------------------|----|----|
|              |           | GT                       | C3PO | GT                       | C3PO | GT               | C3PO | GT                | C3PO | GT                      | C3PO | GT              | C3PO | TP               | FP | FN |
|              |           | Softnapp                 | 5    | 5                        | 5    | 0                | 0    | 5                 | 5    | 5                       | 5    | 5               | 5    | 0                | 0  | 20 |
| Cstealer     | 1         | 1                        | 1    | 0                        | 0    | 1                | 0    | 0                 | 0    | 0                       | 0    | 0               | 0    | 1                | 0  | 1  |
| Ragebot      | 2         | 0                        | 0    | 0                        | 0    | 2                | 2    | 2                 | 2    | 0                       | 2    | 2               | 2    | 6                | 2  | 0  |
| Expiro       | 3         | 0                        | 0    | 0                        | 0    | 3                | 3    | 3                 | 3    | 3                       | 3    | 0               | 0    | 9                | 0  | 0  |
| Sathurbot    | 2         | 2                        | 0    | 0                        | 0    | 2                | 2    | 2                 | 2    | 2                       | 2    | 0               | 0    | 6                | 0  | 2  |
| Icloader     | 1         | 0                        | 0    | 0                        | 0    | 1                | 1    | 1                 | 1    | 1                       | 1    | 0               | 0    | 3                | 0  | 0  |
| Alma         | 2         | 0                        | 0    | 0                        | 0    | 2                | 2    | 2                 | 2    | 2                       | 2    | 0               | 0    | 6                | 0  | 0  |
| Zbot         | 4         | 0                        | 0    | 0                        | 0    | 0                | 1    | 0                 | 0    | 4                       | 4    | 0               | 0    | 4                | 1  | 0  |
| Rbot         | 2         | 1                        | 1    | 0                        | 0    | 2                | 2    | 2                 | 2    | 2                       | 2    | 2               | 2    | 9                | 0  | 0  |
| Slackbot     | 4         | 0                        | 0    | 0                        | 0    | 4                | 4    | 4                 | 4    | 0                       | 0    | 0               | 0    | 8                | 0  | 0  |
| Delf         | 4         | 0                        | 0    | 0                        | 0    | 4                | 4    | 0                 | 4    | 4                       | 4    | 0               | 0    | 8                | 4  | 0  |
| Blackhole    | 3         | 2                        | 1    | 2                        | 2    | 3                | 3    | 3                 | 3    | 3                       | 3    | 0               | 0    | 12               | 0  | 1  |
| Equationdrug | 2         | 0                        | 0    | 0                        | 0    | 2                | 2    | 2                 | 2    | 2                       | 2    | 2               | 2    | 8                | 0  | 0  |
| <b>Total</b> | 35        | 11                       | 8    | 2                        | 2    | 31               | 31   | 26                | 30   | 28                      | 30   | 6               | 6    | 100              | 7  | 4  |

### 3.3.2 C&C Monitoring Capabilities Evaluation

Table 3.5 presents our evaluation of C3PO’s ability to identify C&C monitoring capabilities. Columns 3-8 present the capabilities we consider in our study, but C3PO can be extended to support other capabilities. Their sub-columns are divided into two categories: GT and C3PO represents the number of ground truth capabilities identified and automatically identified, respectively. We found that C3PO correctly identified 100 (TP) C&C monitoring capabilities. Our GT analysis confirmed 104 capabilities, revealing 7 FPs, 4 FNs, and an overall accuracy of over 94%.

Table 3.5 shows that Victim Profiling ranks highest among the capabilities, accounting for 29% (31 of the 104) of the capabilities. Next are Live Monitoring and File Exfiltration, with 28 (27%) and 26 (25%) capabilities, respectively. Toward covert monitoring, this shows that the authorities can expect to locate victim information on the C&C server including system information, personal files, and legally admissible evidence of spying.

Among 35 variants, 3 of them (Ragebot, Zbot, and Delf) had 7 FPs from the Victim Profiling, File Exfiltration, and Live Monitoring identification. Next, 3 of the FNs occurred in the Browser Password Stealer while 1 occurred in the Victim Profiling distributed among Cstealer, Sathurbot, and Blackhole. Further investigation revealed that both the FPs and the FNs are attributed to issues experienced using angr either due to unresolved symbolic constraints during CFG generation or temporary variable reuse causing spurious dependencies in the backward slice. However, our investigation confirmed these are rare occurrences. Given the low number of FPs and FNs, and over 94% accuracy, C3PO provides the means to effectively identify C&C monitoring capabilities.

## **3.4 Large-scale Deployment**

We deployed C3PO to measure over-permissioned protocols and C&C monitoring capabilities. We demonstrate that our automated measurement pipeline provides a scalable means

for over-permissioned bot analysis.

### 3.4.1 Post Deployment Dataset Highlights

Deploying C3PO on our dataset exposed a growing trend of over-permissioned protocol use in malware. C3PO revealed that 62,202 (over 30%) of malware use one or more over-permissioned protocols. Figure 3.4 illustrates the adoption of over-permissioned protocols per bot from April 2006 to June 2020. We found that over-permissioned protocol use peaked in years 2015-2019, which also accounted for 80% of all over-permissioned protocols C3PO identified in our study. Interestingly, Figure 3.4 shows that not only has the use of over-permissioned protocols increased, but also the number of protocols used per malware. While a single bot using multiple over-permissioned protocols was uncommon, this practice is more prevalent now than ever before with over 4,000 bots using multiple protocols. In fact, since 2019 alone, C3PO found over 1,500 malware that used more than one over-permissioned protocol.

C3PO found the remainder of the malware (i.e., 70%) in our dataset used only HTTP-based communication for command and control. The prevalence of HTTP-based communication in our dataset is inline with observations by Peredisci et. al. [109], who reported that 75% of malware exhibit network activity via HTTP-based communication. This preva-

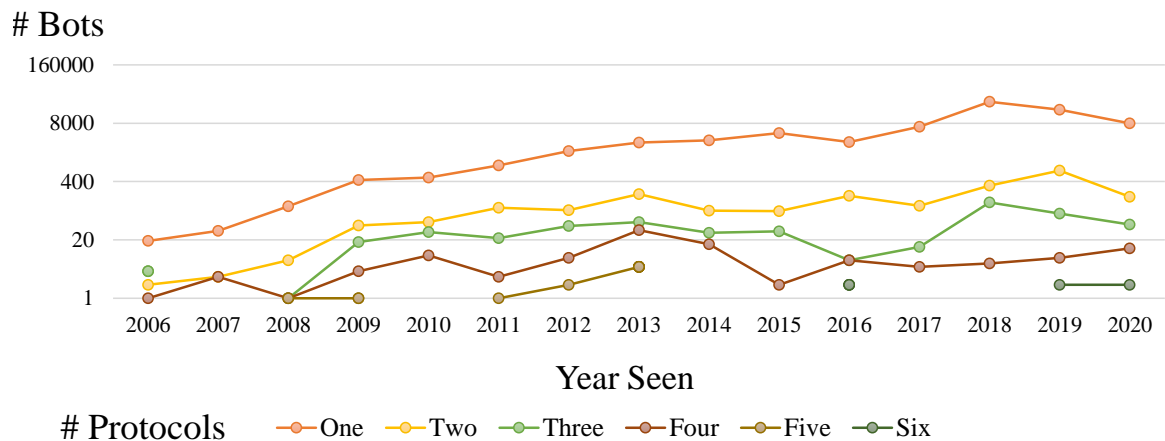


Figure 3.4: Number of Over-Permissioned Protocols Per Bot From 2006-2020.



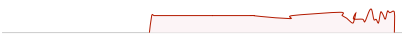






lence led many prior works [110, 111, 112] to target HTTP-based malware exclusively. HTTP-based malware send and receive data in HTTP packets using non-standard message protocols. Unlike the protocols considered in this chapter, these HTTP-based messages do not readily provide authorities access to the C&C server. As such, these HTTP-only malware do not function as over-permissioned bots.

### 3.4.2 Over-Permissioned Bot Landscape

Table 3.6 presents interesting insights into over-permissioned protocol use. Column 1 shows the protocols we studied; Column 2, the number of protocol uses; and Columns 3-4, the total number of  $H_L$  and  $L_L$  identifiers found. Columns 5-6 show their distribution with temporal protocol changes of each sample displayed in Column 8. The total number of malware families observed using specific protocols, as well as the first and last time the malware was seen in 2006-2020 are presented in the remaining columns.

C3PO detected 65,739 uses of over-permissioned protocols. Confirming our hypothesis that protocol efficiency supports continued prevalence, FTP is predominant among all protocols occurring 53,687 accounting for 81% of protocols identified (Column 2, Row 1). Besides, FTP has been consistently used across 88% of the 8,512 malware families over 15 years. This confirms our suspicion that malware authors either do not realize the inherent vulnerabilities from using over-permissioned protocols or simply do not expect them to be used as IVs. We expect the latter to be the case given the known FTP insecurities. Thus, bot orchestrators are unknowingly leaving the front door wide open, a trend our study sheds light on.

Table 3.6: Distribution of Over-Permissioned Bots Identified During the Large-Scale Study.

| Over-Permissioned Protocol | #Used               | #Protocol ID |        | Identifiers/Protocol |     |     | Temporal Changes<br>2006 - 2020   | #Families | First Seen           | Last Seen |
|----------------------------|---------------------|--------------|--------|----------------------|-----|-----|---|-----------|----------------------|-----------|
|                            |                     | $H_L$        | $L_L$  | Min                  | Avg | Max |   |           |                      |           |
| FTP/TFTP                   | 55,494              | 32,531       | 22,963 | 2                    | 4   | 9   |    | 8,163     | 2007-07              | 2020-06   |
| BitTorrent/ $\mu$ TP       | 953                 | 892          | 61     | 1                    | 7   | 21  |    | 56        | 2011-04              | 2020-06   |
| WebDAV                     | 2,963               | 2846         | 117    | 8                    | 16  | 21  |    | 135       | 2012-07              | 2020-06   |
| MongoDB                    | 1                   | 0            | 5      | 5                    | 5   | 5   |    | 1         | 2019-11              | 2019-11   |
| ODBC                       | 670                 | 670          | 0      | 1                    | 3   | 12  |    | 126       | 2010-08              | 2020-05   |
| MySQL                      | 262                 | 262          | 53     | 1                    | 5   | 11  |    | 53        | 2008-12              | 2020-05   |
| PostgreSQL                 | 117                 | 117          | 25     | 2                    | 4   | 7   |    | 25        | 2009-06              | 2019-10   |
| MQTT                       | 24                  | 23           | 18     | 1                    | 2   | 3   |   | 1         | 2014-04              | 2015-12   |
| IRC                        | 10,458              | 0            | 10,458 | 1                    | 3   | 6   |  | 400       | 2006-10              | 2020-06   |
| Total                      | 65,739 <sup>1</sup> | 34,495       | 33,607 | 2                    | 3   | 7   |   | 8,512     | <sup>1</sup> 2006-10 | 2020-06   |

1: This is not the sum of this column, but the total number of protocols uses or malware families (see column 9) detected.



From Table 3.6, Column 8 illustrates the number of protocol identifiers used in malware since 2006. The number of identifiers per sample has generally fluctuated, except for the MQTT and IRC protocols. Similar to the findings in section 3.3, IRC implementations generally use 3 tokens to communicate with the C&C server as illustrated in the average identifiers per use (Column 6). The total distribution of protocol identifiers found (min=2, avg=3, max=7) indicates that many protocols use multiple APIs or tokens giving authorities multiple monitoring vantage points. Notably, the MongoDB protocol was used in the Cstealer malware then spontaneously disappeared from use resulting in a single spike in Column 8. This is likely due to its rapid discovery and public reporting [113], which immediately revealed a weakness resulting from the use of the over-permissioned MongoDB protocol. We expect to see a resurgence of the MongoDB protocol as some malware authors continue to prefer efficiency and ease of use over security. Furthermore, the temporal changes in levels of protocol implementations (i.e., identifiers used) gives us insights into the type of protocol capabilities enabled; e.g., if C3PO identifies `FTPGetFile` or `FTPPutFile` in the malware, the C&C supports at least FTP read/write. However, as our study shows, it is safe to assume full protocol implementation since malware operators adopt over-permissioned protocols for ease of use and scalability.

Table 3.6, Row 8 (*MQTT*) shows a similar trend as the MongoDB protocol. However, the use of the MQTT protocol is observed over a longer period. The Expiro malware is the only malware family detected using the MQTT protocol and disappeared from detection in 2015. Interestingly, industry experts observed and reported on a resurgence of Expiro in 2017 [114] adding clarity to our observation, given that we did not detect it between 2015 and 2020. Industry experts also reported improvements in Expiro, which we believe correlates with its lack of presence in recent years, likely stemming from its exclusion of the MQTT protocol.

From Table 3.6, we also observe that  $L_L$  identifiers are the majority with 33,636 detected versus 33,486  $H_L$  identifiers, although, only by a slight margin. However, the ma-

majority of  $L_L$  implementations resulted from FTP and IRC protocols. As discussed, the IRC protocol has no  $H_L$  implementation. Although many IRC bots are no longer active because of the centralized architecture, which has proven limitations, new IRC malware have been detected in 2020. We now turn our attention to the 8,512 malware families identified. This is important as it illustrates the wide-scale applicability of C3PO across multiple malware families and variants.

### 3.4.3 C&C Monitoring Capabilities at Scale




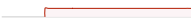


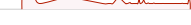











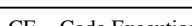
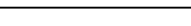
C&C monitoring capabilities guide covert monitoring after C&C server infiltration. As shown in Table 3.7, C3PO identified 443,905 C&C monitoring capabilities in 62,202 over-permissioned bots revealing an average of 7 capabilities per bot. Notably, Victim Profiling and Live Monitoring account for the majority of capabilities, at 56% and 27% with 249,051 and 120,290 identifications, respectively. It follows that the majority of over-permissioned bots use techniques that can be applied more broadly to information stealing, which Victim Profiling and Live Monitoring provide. File Exfiltration is the next commonly used (i.e., 69,041 capabilities), 15% of all capabilities identified.

Of all 16 capabilities, 375,946 or 88% and 43,222 or 10% of them occur in FTP and IRC protocols, respectively. C3PO's ability to extract IVs for these protocols alone allows the authorities to covertly monitor over 85% of over-permissioned bots in our dataset, which we believe is representative of the larger malware landscape. Another observation is that although password stealers capture sensitive victim information, their tactics are tailored for a limited number of applications or services, reducing the scale of their impact explaining the low numbers in the password stealing categories at 5,095 or 0.01%. Overall, C3PO reveals the composition and content of C&C servers through C&C monitoring capabilities identification allowing it to provide targeted monitoring post infiltration.

Table 3.7: C3PO Identification of C&C Monitoring Capabilities Mapped to Over-Permissioned Protocols.

| C&C Monitoring Capabilities      | FTP/FTTP       | BitTorrent/μTP | WebDAV        | MySQL        | MongoDB  | ODBC         | PostgreSQL | IRC           | MQTT       | Total          |
|----------------------------------|----------------|----------------|---------------|--------------|----------|--------------|------------|---------------|------------|----------------|
| <b>Service Password Stealing</b> |                |                |               |              |          |              |            |               |            |                |
| WiFi Stealer                     | 1              | 0              | 0             | 0            | 0        | 0            | 0          | 0             | 0          | 1              |
| Kerberos Stealer                 | 3              | 0              | 0             | 0            | 0        | 0            | 0          | 1             | 0          | 4              |
| Windows Sys. Stealer             | 7              | 0              | 0             | 0            | 0        | 1            | 0          | 1             | 0          | 9              |
| <b>Subtotal</b>                  | <b>11</b>      | <b>0</b>       | <b>0</b>      | <b>0</b>     | <b>0</b> | <b>1</b>     | <b>0</b>   | <b>2</b>      | <b>0</b>   | <b>14</b>      |
| <b>Code Reflection</b>           |                |                |               |              |          |              |            |               |            |                |
| Code Reflection                  | 202            | 0              | 140           | 1            | 0        | 6            | 2          | 77            | 0          | 428            |
| <b>Subtotal</b>                  | <b>202</b>     | <b>0</b>       | <b>140</b>    | <b>1</b>     | <b>0</b> | <b>6</b>     | <b>2</b>   | <b>77</b>     | <b>0</b>   | <b>428</b>     |
| <b>Browser Password Stealing</b> |                |                |               |              |          |              |            |               |            |                |
| Internet Exp. Stealer            | 1,611          | 0              | 6             | 0            | 0        | 1            | 0          | 11            | 0          | 1,629          |
| Chrome Stealer                   | 812            | 0              | 0             | 1            | 1        | 5            | 0          | 5             | 0          | 824            |
| Mozilla Stealer                  | 2,103          | 0              | 496           | 3            | 0        | 2            | 0          | 24            | 0          | 2,628          |
| <b>Subtotal</b>                  | <b>4,526</b>   | <b>0</b>       | <b>502</b>    | <b>4</b>     | <b>1</b> | <b>8</b>     | <b>0</b>   | <b>40</b>     | <b>0</b>   | <b>5,081</b>   |
| <b>File Exfiltration</b>         |                |                |               |              |          |              |            |               |            |                |
| High-level Protocols             | 6,891          | 6              | 1188          | 0            | 0        | 65           | 2          | 60            | 0          | 8,212          |
| Raw Socket Transfer              | 52,223         | 110            | 1168          | 210          | 0        | 510          | 110        | 6,374         | 24         | 60,729         |
| <b>Subtotal</b>                  | <b>59,214</b>  | <b>116</b>     | <b>2,356</b>  | <b>210</b>   | <b>0</b> | <b>575</b>   | <b>112</b> | <b>6,434</b>  | <b>24</b>  | <b>69,041</b>  |
| <b>Spying/Live Monitoring</b>    |                |                |               |              |          |              |            |               |            |                |
| Audio Capture                    | 10,788         | 1              | 62            | 30           | 0        | 15           | 0          | 86            | 0          | 10,982         |
| Keylogger                        | 39,551         | 84             | 2,296         | 185          | 0        | 348          | 113        | 4,256         | 24         | 46,857         |
| Screen Capture                   | 52,458         | 109            | 2,524         | 220          | 0        | 537          | 110        | 6,469         | 24         | 62,451         |
| <b>Subtotal</b>                  | <b>102,797</b> | <b>194</b>     | <b>4,882</b>  | <b>435</b>   | <b>0</b> | <b>900</b>   | <b>223</b> | <b>10,811</b> | <b>48</b>  | <b>120,290</b> |
| <b>Victim Profiling</b>          |                |                |               |              |          |              |            |               |            |                |
| Victim Locale Info.              | 51,924         | 99             | 2,462         | 217          | 0        | 512          | 98         | 6,441         | 24         | 61,777         |
| System OS Details                | 52,354         | 110            | 2,518         | 222          | 1        | 530          | 110        | 6,469         | 24         | 62,338         |
| Registry-stored Info.            | 52,354         | 110            | 2,512         | 225          | 1        | 528          | 98         | 6,471         | 24         | 62,323         |
| Live OS State                    | 52,564         | 110            | 2,510         | 226          | 1        | 534          | 117        | 6,477         | 24         | 62,563         |
| <b>Subtotal</b>                  | <b>209,196</b> | <b>420</b>     | <b>10,002</b> | <b>890</b>   | <b>3</b> | <b>2,104</b> | <b>423</b> | <b>25,858</b> | <b>92</b>  | <b>249,051</b> |
| <b>Total</b>                     | <b>375,946</b> | <b>730</b>     | <b>17,882</b> | <b>1,540</b> | <b>4</b> | <b>3594</b>  | <b>760</b> | <b>43,222</b> | <b>164</b> | <b>443,905</b> |

Table 3.8: Evolution of the Top 10 Families of Over-Permissioned Bots Detected in our Dataset.

| Malware Family | #Over-Permissioned Bots | Over-Permissioned Protocols       | Protocol Use |     |     | Evolution of Protocol Use<br>2006 - 2020  | C&C Monitoring Capabilities <sup>1</sup> | # C&C Monitoring Capabilities |     |     | Evolution of C&C Monitoring Capabilities<br>2006 - 2020                               |
|----------------|-------------------------|-----------------------------------|--------------|-----|-----|---|--|-------------------------------|-----|-----|---|
|                |                         |                                   | Min          | Avg | Max |   |  | Min                           | Avg | Max |   |
| Dinwod         | 9,713                   | FTP                               | 1            | 1   | 1   |    | BPS, VP, FE, LM                          | 3                             | 3   | 4   |    |
| Autoit         | 5,763                   | FTP, IRC                          | 1            | 1   | 2   |    | BPS, VP, FE, LM                          | 3                             | 3   | 4   |    |
| Softcnaapp     | 4,382                   | FTP, IRC, ODBC                    | 1            | 1   | 2   |    | BPS, VP, FE, LM                          | 3                             | 3   | 4   |    |
| Delf           | 4,331                   | FTP, IRC, MySQL, TFTP, ODBC       | 1            | 1   | 3   |    | BPS, VP, FE, LM, CE                      | 3                             | 3   | 5   |    |
| Wabot          | 2,388                   | IRC                               | 1            | 1   | 1   |    | VP, FE, LM                               | 3                             | 3   | 3   |    |
| Fareit         | 1,479                   | FTP, IRC, ODBC                    | 1            | 1   | 2   |    | BPS, VP, FE, LM                          | 2                             | 3   | 4   |    |
| Sivis          | 1,167                   | FTP, IRC, ODBC, MySQL, Bittorrent | 1            | 1   | 3   |  | BPS, VP, FE, LM, CE                      | 3                             | 3   | 5   |  |
| Lamer          | 1,019                   | FTP, IRC, ODBC                    | 1            | 1   | 2   |  | BPS, VP, FE, LM                          | 3                             | 3   | 4   |  |
| Virut          | 998                     | FTP, IRC, ODBC, MySQL             | 1            | 1   | 3   |  | BPS, VP, FE, LM                          | 3                             | 3   | 4   |  |
| Snojan         | 897                     | FTP, IRC                          | 1            | 1   | 2   |  | BPS, VP, FE, LM                          | 3                             | 3   | 4   |  |

1: BPS = Browser Pwd Stealing, VP = Victim ID, FE = File Exfiltration, LM = Live Monitoring, CE = Code Execution (see Table 3.7)

#### 3.4.4 Ranking Over-Permissioned Bot Families

Table 3.8 presents the protocols and the C&C monitoring capabilities identified in the top 10 malware families of our study. The Dinwod malware family ranks the highest with 9,713 over-permissioned bots. Dinwod only uses FTP and has remained consistent, even in the analysis of capabilities that include Browser Password Stealing (BPS), Victim Profiling (VP), File Exfiltration (FE), and Live Monitoring (LM), averaging 3 capabilities per sample.

Another observation from Table 3.8: FTP is used in 9 of the top 10 malware families as expected since it is the most prominent over-permissioned protocol in our dataset. About half of the families in Table 3.8 maintain a generally consistent number of protocols used, with the exception of Delf, Sivis, and Virut, with 3 maximum protocols used each, attributing to the spikes in Column *Evolution of Protocol Use*.

Lastly, the top capabilities — File Exfiltration, Live Monitoring, and Victim Profiling — appear in all 10 families. However, we did not expect Browser Password Stealing in 9 families since it accounts for 1.26% of all C&C monitoring capabilities (ref. Table 3.7). From this study, we can infer that while the majority of over-permissioned bots can be considered Information Stealers, many of the top malware families are Password Stealers.

#### 3.4.5 Packed Malware

C3PO uses a hybrid approach to analyze packed malware (subsection 3.2.1). We present the most common packers encountered in our study in Table 3.9 and use the packer taxonomy proposed by Ugarte-Pedrero et al. [92]. Column 1 lists these packer types, and Column 3 shows the number of packed malware that use the packers presented in Column 2. The packer types range from Type-I to Type-VI, which also represent their order of complexity [92].

For clarity, Type-I packers are the easiest to unpack only using a single unpacking routine before transferring control to the malware payload. Type-II packers use multiple

layers of packing, only transferring control after the last layer is complete. Like Type-II packers, Type-III is multi-layered but does not unpack in a top-down manner and instead uses complex layer organization. While Type-IV packing can use single or multi-layers, the unpacking routine is interwoven with the malware payload switching control back and forth. Type-V and Type-VI are quite similar to Type-IV, except more and more malware payload code is interwoven increasing the complexity of the unpacking routine.

From Table 3.9, we see that C3PO can unpack and analyze samples packed with Armadillo (Row 3), i.e., it can handle the most complex category of packers. Of the 62,202 over-permissioned bots, C3PO unpacked 10,237 malware. The remainder of the samples were not packed. In our dataset, C3PO did not encounter any malware packed with Type-II, Type-IV, and Type V packers. But given its ability to handle Type-VI packers, we believe that C3PO is robust enough to enable a large-scale study of modern malware.

### 3.5 C3PO Applied

We present two over-permissioned bot case studies to illustrate the efficacy of our techniques. We focus on the cases that use the most prevalent FTP over-permissioned protocol. We redact the C&C server information because the servers are still active as of this writing, but present the monitoring outputs we extracted adhering to ethical practices, which we describe next.

Table 3.9: Packers Encountered in our C3PO’s Dataset.

| <b>Packer Type [92]</b> | <b>Packer [115]</b> | <b>#Malware</b> |
|-------------------------|---------------------|-----------------|
| Type-I                  | UPX                 | 9,372           |
|                         | BobSoft Mini Delphi | 86              |
| Type-III                | ASPack              | 48              |
|                         | ASProtect           | 22              |
| Type-VI                 | Armadillo           | 709             |
| <b>Total</b>            |                     | <b>10,237</b>   |

### 3.5.1 Ethical Considerations

We follow the precedence established in previous works [116, 17, 117, 88] while exposing the weaknesses that make C&C servers vulnerable to infiltration. Besides, Burnstein [118] provides legal and ethical conduct for cybersecurity research, arguing that injecting traffic into C&C servers can be considered consent when using the communication channel the bot orchestrators provided to the enslaved systems. Similarly, we use the bot-to-C&C channel and the authentication details provided to us through the malware. Moreover, after verifying access permissions we (1) only retrieve the metadata (e.g. file quantity, table schema, etc.) of the service being investigated (FTP, MongoDB, etc.) and (2) perform no write operations. We emphasize that we *do not* exploit, disrupt, or attempt takedown of C&C servers, avoiding any claim of *tortious interference* as described in Mouton vs. VC3 [119].

### 3.5.2 Case Study 1: Steam

The Steam malware is a Remote Access Trojan (RAT) [120] first discovered in 2016 and persists today. C3PO identified FTP in Steam and extracted IVs and C&C monitoring capabilities (Table 3.10, Rows 1-3). Leveraging the IVs, C3PO covertly monitored the Steam C&C server resulting in the identification of approximately 50 MB of data (522 files

Table 3.10: C3PO’s Steam Malware Analysis Results.

|  |   |
|--|---|
| <b>Protocol</b>                        | FTP   |
| <b>Username:</b>                       | j91{***}  |
| <b>Infiltration Password:</b>          | Dom{***}  |
| <b>Vectors Server:</b>                 | {***}.beget.tech  |
| <b>Port:</b>                           | 21  |
| <b>C&amp;C Monitoring Capabilities</b> | Victim Profiling and<br>File Exfiltration   |
| <b>Covert Monitoring Outputs</b>       | (1) Country of origin, (2) Steam Authentication Files<br>(3) XSS injection script |

in 5 directories). Of the files, 27% of them have “game”-related names like *matchroom* and *tournament* confirming that our sample is indeed tailored for the Steam platform.

C3PO identified Victim Profiling and File Exfiltration, so we expected to find a large number of files on the C&C server containing stolen victim information. Since this malware is relatively new, it is not surprising that we only found less than 20% of these files, but we expect it to grow as the malware spreads. However, C3PO identified two *data* files whose filenames began with *ssfn*. The authorization files for the Steam online gaming platform also begin with *ssfn*. These files are likely encrypted since their entropy values are 7.90 and 7.92 (on a scale of 0.0 - 8.0), respectively. These authorization files could either be stolen files for authentication to the Steam platform (as suggested by C3PO’s File Exfiltration C&C monitoring capabilities), or they belong to the bot orchestrator. For the latter case, an incident responder can use these files to pursue attribution since it provides access to the bot orchestrator’s account.

C3PO also revealed filenames that piqued our interest. Specifically, several files are named in the Russian language, the C&C server’s likely country of origin. Furthermore, C3PO discovered a JavaScript file containing code that looked for cross-site scripting (XSS) vulnerabilities. This suggests further malicious intent to perpetrate additional cybercrimes. Our findings are further confirmed by a Steam analysis report [120] validating C3PO’s effectiveness in covert monitoring and extracting valuable insights.

### 3.5.3 Case Study 2: Detplock

The Detplock malware is another RAT first seen in 2016 and is still active today. This malware allows the bot orchestrator to execute commands on the infected machines. Table 3.11 summarizes C3PO’s covert monitoring results by analyzing the DeptLock malware. C3PO extracted IVs such as the username, password, server address, and port, as shown in Table 3.11, Row 1. Based on the sever address suffix `.ko.kr`, the C&C server is likely located in South Korea. This C&C server responds to FTP queries, which we used to only



catalog file metadata, enumerating directories, keeping count of the number of directories and files as well as file extensions and file sizes. Overall, we identified approximately 640MB of data including over 2,500 files across 47 directories. Of the 31 file extensions found, the most common extensions were PNG (44%), HTML (34%), TXT (8%), and EXE (6%).

C3PO also identified Victim Profiling, Live Monitoring, and File Exfiltration capabilities (Table 3.11, Row 2). From covert monitoring, C3PO discovered many PNG files, which was expected since its analysis showed that Detplock performed Live Monitoring by taking PNG screenshots. This confirms the effectiveness of C3PO’s C&C monitoring capabilities towards covert monitoring. C3PO also located the `userData` directory which is used to store victim information, corresponding to the Victim Profiling malware capability (Table 3.11, Row 3). While this directory was empty upon infiltration, covert monitoring allows us to regularly monitor for added infected systems to understand the scope of infection and enable peer disclosure.

Lastly, C3PO found malicious files on the C&C server’s `download` directory, confirming that Detplock spreads other payloads. Specifically, 7 of the 158 Windows EXE and 2 BIN files contained suspicious metadata. Their signatures revealed ASPack v2.12 packing and their hash search on VirusTotal [121] confirmed maliciousness. Although the C&C

Table 3.11: C3PO’s Detplock Malware Analysis Results.

|  |   |
|--|---|
| <b>Protocol</b> FTP                    |   |
| <b>Username:</b>                       | eg{***}   |
| <b>Infiltration Password:</b>          | vrg{***}  |
| <b>Vectors Server:</b>                 | {***}.co.kr   |
| <b>Port:</b>                           | 21  |
| <b>C&amp;C Monitoring Capabilities</b> | Victim Profiling, Live Monitoring, and File Exfiltration  |
| <b>Covert Monitoring Outputs</b>       | (1) PNG files confirming the live monitoring capability<br>(2) 9 malicious executables and binaries |

monitoring capabilities did not infer additional payloads on the C&C server, our ability to covertly infiltrate and leverage over-permissioned FTP functionality to quickly query the server revealed at least additional 9 malicious files.

### **3.6 Discussion and Limitations**

#### 3.6.1 Domain Generating Algorithms

DGA-based malware allows bot orchestrators to move from centralized architectures to more robust architectures using automatically generated pseudo-random C&C domain names [122]. This technique allows over-permissioned bot orchestrators to subvert persistent infiltrations through C3PO since the C&C domain names are dynamically generated. Other malware adopts a similar approach, using cloud-based services to retrieve C&C domain names [123]. These categories of malware pose significant challenges for C3PO. However, they are not insurmountable, as C3PO can be used to complement existing techniques to identify DGA future candidate domains, as demonstrated by Le et al. [7].

#### 3.6.2 Subverting Dynamic Memory Image Extraction

C3PO's primary technique for memory image extraction is API hooking. As an automated pipeline, C3PO is limited in its ability to spoof specific environments for malware but could be combined with techniques such as forced execution to overcome this [51, 52]. However, sandboxes can also be used to augment C3PO in-lieu of memory image extraction. For example, S2E [46] enables symbolic execution within a sandbox to explore thousands of system paths. Toward unpacking, there are three evasion types to thwart API hooking: stolen code, child process, and process hollowing, often seen in the Themida, PEP, and Pespun packers [91]. Although C3PO can handle Type-I, II, III, and IV packers, it cannot analyze malware that uses virtualization packed techniques. These packers convert programs into bytecode increasing complexity and eludes C3PO memory image extraction. However, virtualization packers account for less than 2% of packed malware, while Type-I

packers (e.g., UPX) account for over 55% [92].

### 3.6.3 Custom Low-level Protocol Implementations

Some malware prefer custom protocol implementations to make their analysis more difficult, but the uniqueness of custom protocols supports signature development increasing their chances of IDS detection. So, C3PO focuses on protocol implementations that adhere to official protocol specifications. However, since C3PO relies on official APIs and tokens, custom tokens evade C3PO's identification. Even if we consider well-known (but not official) tokens, since C3PO analyzes the client-side binary alone, it cannot match a custom keyword to a protocol without knowing how the server parses it. While malware authors can use official protocol commands to trick analysts in misidentifying the protocol used, we have not observed this practice during our large-scale study. In order to support the identification of over-permissioned custom protocols, the integration of tools such as Prospex [32] can be used to automatically reverse engineer custom protocols revealing identifiers that can be *exploited* for covert C&C server monitoring. Although extracting relevant information from the protocol, then adding them to C3PO's protocol database requires some effort upfront, maintenance is all that is required after allowing seamless integration into C3PO.

## **3.7 Conclusion**

This chapter presented C3PO, a measurement pipeline that studied the evolution of over-permissioned protocols in 200k malware spanning back 15 years and how they can be leveraged to provide covert C&C server monitoring. C3PO identified 62,202 over-permissioned bots across 8,512 families identifying infiltration vectors that allow C3PO to spoof bot-to-C&C communication. C3PO also identified 443,905 C&C monitoring capabilities which reveal the composition and contents of the C&C server to guide monitoring post infiltration. We deployed C3PO on two bots with live C&C servers validating its ability to identify over-permissioned protocols, infiltrate C&C servers, and leverage C&C monitoring capa-

bilities to achieve covert monitoring. Furthermore, C3PO identified over 2500 files, some of which contain victim information, additional malicious payloads, exploitation scripts, and stolen credentials providing legally admissible evidence to engender attempts of botnet disruptions and takedowns.

## CHAPTER 4

### R2D2: IS THAT MALWARE READING TWITTER? TOWARDS UNDERSTANDING AND PREVENTING DEAD DROP RESOLVERS ON PUBLIC WEB APPS

Modern malware dynamically resolve their C&C server addresses to be more resistant to disruption or takedown attempts. Prominent examples of these dynamic resolution techniques include fast-flux networks [124, 125, 126], domain name service (DNS) calculation [127, 128], and domain generation algorithms (DGA) [129, 122, 130]. However, fast-flux networks and DNS calculation make C&C servers susceptible to blacklisting [125, 128]. Thus, DGAs preferred mainly as they allow malware to generate C&C server domains dynamically. However, researchers have counteracted DGA-based malware by generating future candidate domains to sinkhole the botnet [7, 4, 5, 18]. As successes in disrupting and taking down DGA botnets increases, malware authors are adopting an under-explored technique: *using dead drops to hide C&C server rendezvous points* (i.e., dead drop resolvers).

Dead Drop Resolvers (DDRs) are posts on public website applications (web apps) that hide C&C server rendezvous points (i.e., domain names, IPs, etc.). Abused web apps include everything from social media networks (e.g., Twitter) to data hosting platforms (e.g., Dropbox) to Bitcoin transactions on the blockchain. These platforms allow anonymous access to user-created content and network traffic to these web apps seems benign, making firewall filtering almost impossible [131, 132]. Unlike DGA-based malware, DDRs protect C&C servers from discovery during malware binary analysis and allow malware authors to migrate their C&C server to truly unpredictable domains with a single post. Worse still, the adoption of DDR techniques leads many malware authors to *encode* C&C rendezvous points before posting on web apps, so viewers and authorities will be unaware of the true

intent. Thus, even when they are identified, authorities must undergo extensive analysis to understand encoding types to decode the C&C domain before any action can be taken. Noting these benefits, more malware authors are transitioning to DDR.

DDR-based malware has created a lop-sided effort between attackers (bot orchestrators) and defenders (authorities). C&C orchestrators need access to a single web app account to hide their C&C rendezvous point. However, defenders (1) need to be able to identify malicious traffic to and from web apps, (2) identify malware data encoding types to help security analysts deduce malicious intents and design defensive solutions, and (3) use the information gathered in hopes of requesting cooperation from service providers before the C&C orchestrator migrates their server to a new domain. With thousands of web apps to choose from, attackers need minimal effort to integrate DDRs into their malware, forcing defenders into additional and often impeding coordination efforts toward botnet disruption or takedown attempts.

To understand the complexities in identifying DDR-based malware, the scope of web app abuse by DDRs, and commonalities among DDR designs, a comprehensive study is needed. Through our collaboration with Netskope, a network edge security provider serving over 25% of the Fortune 100, we studied 100k malware samples captured between 2017 and 2022. Our analysis is based on an automated framework, named R2D2<sup>1</sup>, which uses concolic execution to identify DDR capabilities and their decoding algorithms in malware binaries.

Based on our study, we uncovered 10,170 DDR-based malware from 154 families. Our empirical measurement revealed Pastebin to be the most popular abused web app. To protect their C&C URLs, we found that String Parsing Base64 is the most common encoding in 75% of DDR-based malware. To compound their protection, some malware use multiple encoding techniques to thwart manual decoding, which R2D2 identified in 78% of DDR-based malware. R2D2 includes a novel and generic methodology for recovering the

---

<sup>1</sup>R2D2: **R**evealing **R**endezvous **P**oints from **D**ead **D**rops

DDR decoding *recipe* from malware to enable authorities to decode new C&C rendezvous points when discovered quickly. We reported all of our findings to the appropriate web app providers. Based on our reporting, web app providers confirmed our findings and took action against 9,155 DDRs (90% of our total findings). Of the remainder, the accounts for 774 malware were already taken down at the time of our study. We are awaiting a response from other web app providers hosting the other 241 DDRs.

## 4.1 Overview

Web apps that allow users (malware authors) to post public data under the guise of normal operations are uniquely suited for DDRs. These web apps are not attacker-owned, and some do not verify authentic user identity. A list of DDR web apps found in this study is shown in Table 4.8. This list does *not* include web servers that have been exploited and host malicious content [133, 134].

Over the last few years, industry experts have sounded the alarm on malware abusing web apps [132, 81, 135, 136]. Yet, mitigating these threats has been ineffective. In fact, MITRE suggests traditional intrusion detection systems for either (1) blocking malicious traffic to benign websites, presupposing that authorities can identify malware-specific traffic to benign web apps from all other benign traffic or (2) restricting all access (benign or not) to web-based content used by malware [137]. Given the analysis of the few discovered cases of DDR-based malware [138, 139, 140, 136], these are plausible approaches but only apply to the malware under analysis. Furthermore, as web apps are becoming popular tools for organizations globally [81], blanket restrictions are problematic.

### 4.1.1 Running Example - Razy

*Razy* is a Remote Access Trojan that spawns two threads to repetitively flood VirusTotal with submissions and connect to Twitter to retrieve a message. A previous analysis of *razy* [141] used API call monitoring and network trace analysis to identify what domains

were being contacted. However, the investigators could not confirm how the malware used the message from Twitter, so they instead focused on its VirusTotal-related actions, which only served to distract analysts.

Conversely, R2D2 provides insights into *razy* shedding light on how it uses Twitter to resolve its C&C rendezvous point. R2D2 dynamically executes *razy* and hooks all network-related functions to reroute them to our functions models. R2D2 extracts the web app domain (i.e., `http://www[.]twitter[.]com/pidoras62`) from network connection APIs then backtracks to identify its origin (subsection 4.2.1). This is essential when considering hybrid DDR+DGA techniques in malware. Specifically, many malware hard-code domains or store them in dropped files on the victim system [21], but some DDR-based malware use DGAs to dynamically generate web app account URLs [138]. However, in this case, R2D2 found *razy*'s Twitter domain hard-coded into the binary.

Next, R2D2 continues exploration to confirm DDR (subsection 4.2.2). R2D2 taints the memory region corresponding to the API argument stated to receive data from the web app. When R2D2 arrives at a subsequent network connection, if the domain name parameter is tainted, R2D2 has confirmed DDR integration in *razy*.

Although the previous *razy* investigation could not prove how the malware used the Twitter message, they navigated to the account page and selected the suspicious-looking message<sup>3</sup> (Figure 4.1). An experienced analyst can infer a customized form of Base64 encoding. As R2D2 revealed, *razy* selectively removes preceding characters to find the exact encoded string. After decoding, the tweet resolved to a hacking forum that can be used to send C&C commands to *razy* bots. However, investigators have no way to automatically identify encoding types, and when faced with multi-encoding, even experienced analysts will find decoding a complex task.

In contrast, by symbolizing the data received from Twitter, R2D2 continues to prop-

---

<sup>2</sup>Twitter now only allows `https` requests meaning that older versions of *razy* cannot access the tweet. R2D2's API models allow analysis irrespective of endpoint liveness.

<sup>3</sup>There are only 4 messages posted to date.





Figure 4.1: Twitter Message Retrieved by Razy.

agate the DDR tag through the decoding function. During propagation, the symbolic expression grows according to decoder algorithm computations. R2D2 then uses symbolic expression matching to identify decoders in malware. In *razy*, R2D2 identified that it decodes the Twitter message using String Parsing and Base64. Investigators are now armed with a decoding *recipe* to reveal the current and future *razy* rendezvous points.

The run time for R2D2 to analyze *razy* was 148 seconds. At this rate, R2D2 would provide investigators with rapid results allowing them to quickly distinguish *razy* web app traffic from benign traffic. R2D2 also provides proof that *razy* is abusing Twitter to host C&C rendezvous points. Using this, authorities can request collaboration with web app providers toward counteraction. Web app providers are best equipped to secure their platforms, and they can enable authorities to monitor DDR botnets based on the information R2D2 provides. Furthermore, web app providers can support sinkholing requests, allowing authorities to replace the malware's Twitter messages with their own, *encoded in the correct format*, resulting in similar disruption and takedown to DGA botnets. R2D2's ability to rapidly analyze and report DDR behavior cancels the lopsided effort between attacker and defender.

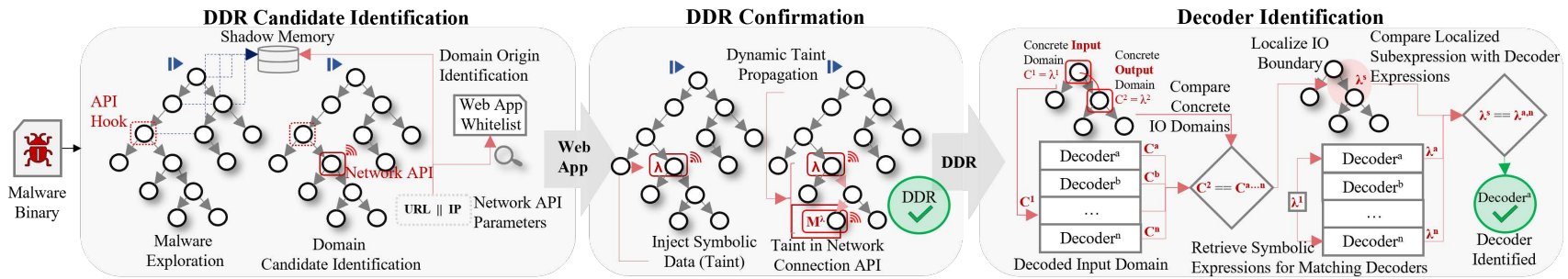


Figure 4.2: R2D2 Measurement Pipeline: Taking a malware binary, C3PO uses *DDR Candidate Identification* via concolic exploration to reveal connected web apps. C3PO then uses concolic taint propagation toward *DDR Confirmation*. After confirming DDR capability, C3PO conducts *Decoder Identification* by concretely localizing decoders in the malware before confirming via symbolic expression matching.

## 4.2 Design

Figure 4.2 shows an overview of R2D2’s pipeline. Taking a malware binary as input, R2D2 identifies dead drop domain candidates by exploring the malware and tracking its execution as it establishes a network connection (subsection 4.2.1). Next, R2D2 induces selective exploration through symbolic data injection to mutate branch predicates allowing the exploration of additional paths. This enables R2D2’s concolic taint propagation to confirm that data received from the web app is used to establish a subsequent outbound connection, i.e., retrieving a hidden C&C server rendezvous point (subsection 4.2.2). Lastly, given that DDR malware use decoders for deobfuscation, R2D2 identifies the type of decoders in malware which provides authorities with the decoding *recipe* to quickly analyze retrieved data (subsection 4.2.3). In total, R2D2 provides the requisite data to glean insights into the emerging DDR-based malware enabling rapid counteraction by authorities.

### 4.2.1 Dead Drop Resolver Candidate Identification

DDR-based malware can retrieve web app domains from the malware itself, a file on the victim system, or dynamically generate them. So, we (1) explore the malware to evaluate connection targets for dead drop candidacy, i.e., web app connection, and then (2) identify its origin.

#### ***Malware Exploration***

Malware often employs packing and obfuscation techniques as well as evasive behaviors that inhibit malware exploration [92]. We also expect some malware network endpoints to be unavailable if they have been disabled. We opt for dynamic analysis paired with symbolic execution (concolic execution) to address these challenges. Inspired by prior works [91, 60, 21, 46], this approach allows the malware to unpack and deobfuscate itself enabling R2D2’s exploration. R2D2 also uses API hooking for symbolic data injection to

simulate network connection and bypass malware evasive techniques. However, although injecting symbolic information exposes more exploration paths, we risk path explosion. We address these challenges below.

**Simulating Victim Systems.** To simulate a victim Windows system, R2D2 executes the malware in QEMU [142] and injects a custom DLL into the malware process, allowing API hooking, which redirects malware API calls to API models. All network-related APIs (Winsock [143], WinInet [144], and WinHTTP [145]) are modeled and hooked to simulate network connectivity. We also model file operation APIs (e.g., `WriteFile`) to trick the malware in the absence of these files. There are several other APIs that malware often uses for anti-analysis or defensive evasion, e.g., `GetLocaleInfo`, which retrieve the location of the victim system before proceeding with the system exploitation. Hooking these API allows us to inject symbolic data into the malware enabling R2D2 to explore all possible paths that follow. The complete list of defensive evasion APIs considered in designing R2D2 is listed in Table 4.1. These APIs are based on our manual malware analysis, reports from industry experts [82, 100], and prior research that investigates victim system emulation and malware evasion techniques [146, 147, 148]. However, R2D2 can be extended to support other operating systems.

**Path Explosion.** Although path explosion is a common problem, R2D2’s exploration is limited to the initial network connections where DDR integration occurs. We expect symbolic loops during R2D2’s execution, especially when decoding functions are called. However, loops often do not lead to increased code coverage [149], so R2D2 limits the number of new states by prioritizing those that lead to unexplored code. To do this, R2D2 maps explored regions and references this map when selecting a new state. Furthermore, based on previous work which discovered that most malware samples run for less than 2 minutes or more than 10, and 98% of the basic blocks are executed within the first 2 minutes [146], we set an upper bound run time to 15 minutes which we find more than sufficient for our study. In fact, for R2D2’s evaluation, we calculated an average run time

Table 4.1: Defensive Evasion APIs Considered in R2D2.

| <b>Anti-Analysis and Defensive Evasion APIs</b> |                           |
|---|---------------------------|
| CheckRemoteDebuggerPresent                      | CreateProcessInternal     |
| CreateProcessWithLogon                          | CreateProcessWithToken    |
| EnumDeviceDrivers                               | EnumDisplayMonitors       |
| EnumServicesStatus                              | FindClose                 |
| FindWindow                                      | GetCursorPos              |
| GetDC   | GetDeviceCaps             |
| GetDiskFreeSpacev                               | GetEnvironmentStrings     |
| GetFileAttributes                               | GetFileSize               |
| GetFileTime                                     | GetKeyboardLayout         |
| GetKeyboardType                                 | GetLastInputInfo          |
| GetLocaleInfo                                   | GetLocalTime              |
| GetOEMCP  | GetServiceKeyName         |
| GetSysColor                                     | GetSystemInfo             |
| GetSystemMetrics                                | GetSystemTimeAsFileTime   |
| GetSystemTimes                                  | GetThreadLocale           |
| GetTickCount                                    | GetTickCount64            |
| GetUserDefaultUILanguage                        | IsDebuggerPresent         |
| IsDebuggerPresentPEB                            | IsProcessorFeaturePresent |
| NtCreateFile                                    | NtGetContextThread        |
| NtGetTickCount                                  | NtEnumerateKey            |
| NtEnumerateValueKey                             | NtOpenFile                |
| NtOpenKey                                       | NtQueryAttributesFile     |
| NtQueryPerformanceCounter                       | NtQuerySystemTime         |
| QueryPerformanceCounter                         | QueryInterruptTime        |
| NtQueryValueKey                                 | RegCloseKey               |
| RegGetValue                                     | RegOpenKey                |
| RegQueryValue                                   | RtlTimeToSecondsSince1970 |
| SetTimer  | Sleep                     |
| SleepEx   | timeGetSystemTime         |
| timeGetTime                                     | timeSetEvent              |

of 375 seconds to identify DDR-based malware.

**Dead Drop Resolver Domain Candidates.** Deducing domain origin requires backtracking to identify where it was constructed, retrieved, or generated without prior knowledge. When the malware invokes a network connection API (e.g., `connect`), R2D2 evaluates

the domain name. If it is an IP address, R2D2 translates IP addresses to domain names via 14.3 billion domain records provided by WhoisXML API. The domain name is then compared against a pre-defined white list of DDR (web app) candidates. This list is based on Tranco [150]<sup>4</sup>, which allows us to identify only benign websites. However, other web apps can be easily included in R2D2 by updating the list.

### ***Dead Drop Resolver Domain Origin***

The 3 categories of origins are: domains that are hard-coded in the malware, (2) retrieved from files located on the victim system, or (3) dynamically generated web app accounts, like DGA-based malware. While (1) and (2) are expected, given that C&C domains are often stored using these methods [21], (3) sheds light on this hybrid DDR+DGA practice which provides authorities insights into counteraction methods against DDR botnets, similar to those employed against DGA botnets. To identify domain origin, R2D2 maintains a shadow memory stack to search for domain indicators.

**Finding Domain Indicators.** When the malware invokes a network connection API, R2D2 extracts the memory location where the domain is stored before searching for indicators in the stack. To identify category (1), R2D2 traces back through the stack to find which instruction last defined the domain memory location. This process recursively occurs until no more definers are found. In this case, R2D2 ends at a concrete value representing the original location of the domain in the malware. While category (2) is solved in the following section (subsection 4.2.2), since it regards symbolic tags that propagate to a network connection API, R2D2 does not explore the path that follows as no actual domain can be retrieved since the file does not exist. However, R2D2 continues its exploration to identify any potential hard-coded backup domains.

Category (3) is similar to (1) in that a portion of the requested URL is hard-coded in the malware, while the account is dynamically generated (e.g., `www.twitter.com` and

---

<sup>4</sup>Available at <https://tranco-list.eu/list/J49Y>.

1b0xsrs). To dynamically generate the web app account name, DGAs use a seed to initialize creation. This seed is often based on a system-available value (e.g., `GetTickCount`), as seen in the popular Conficker DGA malware [151]. Since R2D2 hooks these system query APIs and injects symbolic data to avoid defensive evasion, the portion of the URL corresponding to the account name will be symbolic, while the domain name will be concrete. When R2D2 identifies this symbolic/concrete data, it originates from a DGA.

Notably, current techniques to counteract DGA-based malware are effective [122, 7]. However, they do not provide what is needed for DGA domain origin analysis. Our approach will not improve the effectiveness of DGA counteraction methods but could help simplify their approach in identifying DGAs in malware. In fact, the location of the last API used to generate the DGA seed pinpoints the location of the DGA in the malware.

#### 4.2.2 Dead Drop Resolver Confirmation

After identifying dead drop candidates, R2D2 must identify if data received from the web app is used to establish a subsequent outbound connection. We could inject concrete data into the buffer slated to receive information from the web app, but since DDR-based malware often retrieves encoded data, if the concrete data injected is not formatted as the malware expects, execution can fail and prevent R2D2 from continuing its analysis. To enable DDR confirmation, we use concolic taint propagation to monitor how data propagates throughout malware execution, then trigger an alarm when the taint is used in a network connection API. A common challenge faced with this approach is under-tainting, where control dependence breaks taint propagation (i.e., tainted input only propagates via data dependence or explicit flows). To address this challenge, we stabilize our concolic taint propagation approach to identify the cause, implicit flows, then propagate the tag through control dependence.

### *Concolic Taint Propagation*

To enable concolic taint propagation, we identify a set of APIs to hook so R2D2 can inject symbolic data. These APIs read data from a network resource (e.g., `recv`, etc.). When these APIs are invoked, R2D2 symbolizes the buffer slated to receive data from the web app and attaches a tag (taint) corresponding to the name of the API (e.g., `recv1`). As exploration continues, this tag is propagated with symbolic data. When R2D2 reaches a network connection API, R2D2 checks the argument containing the server name (e.g., `pswzServerName` for `WinHttpConnect`). If the argument is tainted, R2D2 has identified a DDR-based malware. However, if there are implicit flows, R2D2 loses track of the tag. We use a heuristic to identify implicit flows and propagate the tag.

**Implicit Flow Propagation.** When implicit flows cause taint propagation to halt, the malware still executes to its pending network connection. However, R2D2 will be unable to confirm DDR integration since the tag has not propagated. When the malware invokes the network connection API, the server name parameter will likely contain garbage data (since it was based on unconstrained symbolic data). Since R2D2 has already identified the DDR domain candidate, we assume a second connection is based on data received from the web app. To confirm, we must first identify implicit flow blocks then propagate the tag.

An implicit flow block contains source operands that are tainted, but after execution of that block, its succeeding block’s source operands are not tainted. For an illustrative example, refer to Algorithm 1. The input  $a$  to  $Foo$  is tainted (line 1). When line 3 is executed, R2D2 recognizes this as a conditional branch, and since it uses symbolic execution, it knows to fork 2 states corresponding to the branch predicates (lines 4 and 6). R2D2 creates a pair of the branch condition (line 3) and the taint, i.e.,  $\langle 3, t_a \rangle$ , and stores it in a stack. When line 4 is executed, since the stack is not empty, R2D2 peeks the stack value to retrieve  $t_a$  and associates it with  $x$  (line 4). When line 8 is executed, it is not dependent on the conditional branch, so R2D2 pops the stack. Since  $t_a$  has been associated with  $x$ , when



---

**Algorithm 1** Implicit Flow Propagation

---

```
1: function FOO(int a)
2:   int x, y
3:   if a > 10 then
4:     x = 1
5:   else
6:     x = 2
7:   end if
8:   y = 5
9:   print(x)
10:  print(y)
11: end function
```

---

line 9 is executed,  $x$  is correctly tainted. R2D2 uses this approach to identify implicit flow blocks and propagate tags to confirm DDR integration in malware.

#### 4.2.3 Decoder Identification

To identify the decoder type, R2D2 takes the tainted symbolic data (symbolic expression) (subsubsection 4.2.2) and compares it with another symbolic expression representing a known decoding algorithm. However, two questions arise: (1) How do we compare two decoder symbolic expressions? (2) How do we locate the segment of the malware expression that pertains to decoding for an accurate comparison? We answer both of these questions below.

#### *Symbolic Expression Matching*

R2D2 uses Algorithm 2 which takes 2 symbolic expressions  $E^{1\lambda}$  and  $E^{2\lambda}$  as input and returns a ratio of matches versus nonmatches. To generate those expressions, we use R2D2 to symbolically explore each pair of algorithms serially. For example,  $E^1$  is explored with the symbolic data  $\lambda$  as input resulting in  $E^{1\lambda}$ . Then,  $\lambda$  is again used as input to  $E^2$  resulting in  $E^{2\lambda}$ . However, after  $E^1$  is executed,  $\lambda$  assumes the constraints from  $E^1$ 's execution before being used as input to  $E^2$ . Now, when  $E^2$  is symbolically explored, concretized values assumed during forking correspond to the previous constraints imposed by  $E^1$ .

Algorithm 2 has two conditional constructs: (1) check if the overall expressions match, if not (2) solve for and compare the concretized output. Toward clarity, we use a symbolic expressions with starting  $\lambda$  values (`Read byte_1, v0_xor_0`). This expression is used as input to 2 versions of an XOR by `0x23` algorithm and the As the expression is transformed via decoding, it grows with additional operations corresponding to algorithmic computations. For example, 1 byte of  $E^{1\lambda}$  and  $E^{2\lambda}$  is partially transformed into the following:

```

 $E^{1\lambda} = (\text{Or } (\text{ZExt } (\text{Read byte}_1, \text{v0\_xor}_0)) \text{ 0x23})$ 
 $E^{2\lambda} = (\text{Xor } \text{0x23 } (\text{ZExt } (\text{Read byte}_1, \text{v0\_xor}_0)))$ 

```

Then,  $E^{1\lambda}$  and  $E^{2\lambda}$  are compared. This first comparison (lines 4-9) considers node placement, edges, and the size of the expressions. This check is static and less computationally expensive. Thus, it is preferred.

If the first check fails, R2D2 invokes the symbolic solver (lines 10-19) to compare the concretized output of both expressions. When each expression’s child expression results in the same concretized output (line 20), then the expressions are equivalent. These values are based on the symbolic constraints from  $E^1$  that are then imposed during the execution of  $E^2$ . This ensures that if the expressions are the same, they are evaluated based on the same constraints resulting in the same concrete output which confirms their equivalence. A detailed evaluation of this approach is provided in subsection 4.3.2 and Table 4.6.

Using Algorithm 2, R2D2 can match a decoding algorithm symbolic expression to a decoder in malware. However, R2D2 must first identify the portion of the malware expression that corresponds to decoding, which we discuss next.

### ***Localizing Malware Decoders***

We integrate 10 standard decoding algorithms used in malware [82, 100] into R2D2 and list them in Table 4.2. Authorities can add new decoder algorithms by adding their source

---

**Algorithm 2** Symbolic Expression Matching

---

**Input:**  $E^{1\lambda}, E^{2\lambda}$ **Output:**  $\text{match} \div (\text{nomatch} + \text{match})$ 

---

```
1: nomatch = 0
2: match = 0
3: function SYMBOLICMATCHING( $E^{1\lambda}, E^{2\lambda}$ )
     $\triangleright$  Evaluate matching based on symbolic expressions of each algorithm
     $\triangleright$  1st static comparison: one-for-one match
4:   if  $E^{1\lambda} \equiv E^{2\lambda}$  then
5:     match + = 1
6:     return
7:   else
8:     nomatch + = 1
9:   end if
     $\triangleright$  2nd comparison if the previous fails. This comparison invokes the solver to compared the
    concretized output values.
10:  for each  $child_1 \in E^{1\lambda}$  do
11:     $e_{1c} = child_1$ 
12:    for each  $child_2 \in E^{2\lambda}$  do
13:       $e_{2c} = child_2$ 
14:      if  $\neg \text{compConcretizedResults}(e_{1c}, e_{2c})$  then
15:        nomatch + = 1
16:        return
17:      end if
18:    end for
19:  end for
20:  match + = 1
21:  return
22: end function
```

---

Table 4.2: Common Malware Decoding Algorithms.

---

| <b>Decoder Classes</b>                   |
|--|
| Exclusive OR (XOR) [152, 153, 154, 155]  |
| String to Integer [156]                  |
| Integer to String [156]                  |
| Character Rotation [154, 153]            |
| Character Subtraction [154]              |
| Base 16 [138, 139, 154]                  |
| Base 32 [157]                            |
| Base 64 [141, 153, 154, 138, 140]        |
| Base 85 [158, 159]                       |
| String Parsing [141, 156, 153, 160, 139] |

---

code implementation to R2D2. R2D2 will automatically create symbolic expressions for the code to be used during matching.

To identify decoders in malware, R2D2 compares malware and decoder algorithm ex-

pressions. This is challenging because the malware expression includes additional information that is unrelated to decoding. For example, after data is read from a web app, it may contain additional characters used as string markers, or portions of the data may be encoded differently. If we attempt to match a decoder algorithm expression to an entire malware expression, we will likely generate a false negative result where we miss identifying a decoder. Unlike a direct comparison of like expressions, R2D2 must first localize the portion of the malware expression that relates to decoding. Only then can R2D2 use Algorithm 2 on a localized subexpression to confirm the use of any decoding algorithms.

**Decoder Localization.** Confirming DDR-based malware relies on symbolizing the buffer that stores data read from the web app (subsection 4.2.2). As the symbolized malware buffer, referred to as  $M^\lambda$ , is decoded, it assumes operations corresponding to data computations. However, without identifying the segment of  $M^\lambda$  that corresponds to decoding, R2D2 will not be able to match it with any decoder algorithms. So, R2D2 localizes the boundary around the decoding algorithm via concrete input/output (IO) domain matching. This is done using Algorithm 3 which occurs in conjunction with DDR Confirmation (subsection 4.2.2). We use the malware as input to the algorithm to reflect this point. When complete, Algorithm 3 returns a set of all confirmed decoders identified.

**Input/Output Domain Mapping.** Throughout R2D2’s malware exploration (Algorithm 3, lines 3-4), it tracks memory address accesses (line 6). This allows R2D2 to keep track of  $M^\lambda$  as it is moved through memory<sup>5</sup>. Every time  $M^\lambda$  is accessed, R2D2 calculates concrete values for each byte of the buffer (line 7) for input to all decoder algorithms and stores the output resulting in a map of data including the current instruction, concretized  $M^\lambda$ , and the decoding results from each decoding (lines 8-11). However, R2D2 also compares live execution results with previously-stored results to localize the decoding boundary (lines 12-18). If a previously decoded result at instruction  $A$  matches to a concretized memory value at instruction  $J$ , then the boundary is  $A - J$ .

---

<sup>5</sup>We also consider when it is moved from memory to registers.

---

**Algorithm 3** Malware Decoder Identification

---

**Input:**  $M, DI = [\{D_a^{1\lambda} \dots D_a^{n\lambda}\}, \dots \{D_z^{1\lambda} \dots D_z^{n\lambda}\}]$ **Output:** *confirmedAlgos*

---

```
1: decMap = NULL
2: algoBoundary = NULL
3: function LOCALIZEDECODERBOUNDARY(M)
4:   Inst = stepInstruction(M).successor(
                                     ▷ Explore malware by insrtuction-by-instruction
5:   while stepInstruction(M) do
                                     ▷ At each symbolic memory access, extract the symbolic data and concretize it
6:     if mem = memoryAccess(inst) then
7:       concrete = concretizeMem(mem)
                                     ▷ Use the concrete values as input to all decoders and save the results
8:       for each  $D^\lambda \in DI$  do
9:         decoded =  $D^\lambda(\textit{concrete})$ 
10:        decMap[Inst] = {concrete $_{D^\lambda}$ , decoded $_{D^\lambda}$ }
11:       end for
                                     ▷ To identify the decoder boundary, match the current concrete values with previously decoded
                                     results and compute the distance between
12:       for each element  $\in$  decMap do
13:         if decoded = element.concrete $_{D^\lambda}$  then
14:           instTuple = (Inst, element.Inst)
15:           d = Distance(instTuple)
16:           algoBoundary[ $D^\lambda$ ] = [(d, (instTuple))]
17:         end if
18:       end for
19:     end if
20:     Inst = stepInstruction(M).successor()
21:   end while
22:   confirmedAlgos = NULL
                                     ▷ Confirm decoders with symbolic equivalence testing based on the shortest distance
                                     sub-expression
23:   for each algo  $\in$  algoBoundary do
24:     start, end = minDistance(algo)
25:     subExp =  $M^\lambda[\textit{start}, \textit{end}]$ 
                                     ▷ Call to Algorithm 2 to compare expressions and store matches if found
26:     match = SymbolicEquivalence(subExp, algo.D $^\lambda$ )
27:     if match then
28:       confirmedAlgos[ $D^\lambda$ ] = (start, end)
29:     end if
30:   end for
31:   return confirmedAlgos
32: end function
```

---

Matching the IO domains with concrete execution of decoder algorithms may seem sufficient, but when two decoders produce the same result, the matching would be inconclusive. For example, if 0x5956645762467059546e633d is decoded with Base64, the result is 0x6157566c5a584e77. If we use the same input and XOR it with

0x59566457031126350E362D4A, we arrive at identical outputs. If the C&C orchestrators post additional encoded messages, relying on only the IO boundary could lead authorities to apply the incorrect decoder. Therefore, using the boundary, R2D2 matches expressions to confirm decoders.

**Sub-Expression Decoder Comparison.** R2D2 identifies multiple decoder boundaries when the memory region goes unmodified for a portion of malware execution. Although R2D2 would be able to perform symbolic expression matching at these larger boundaries, by computing the distance between instructions corresponding to matching IO domains, R2D2 can more precisely locate the decoding function in the malware (line 15). Iterating through the identified boundaries, R2D2 takes the minimum distance match and uses the start and end instruction values to extract a subtree expression from  $M^\lambda$  (lines 22-25). Next, the subexpression tree and the expression from the decoder algorithms are compared for equivalence (line 26). We generate symbolic expression for all decoding algorithms and store them for future referencing by R2D2 during its execution. If there is a confirmed match, R2D2 reports the identified decoder in the malware.

After taking a malware binary as input, R2D2 reports if the malware integrates DDR capabilities and what web app it relies upon. R2D2 also reports the decoder(s) used to encode the C&C rendezvous point allowing authorities the ability to reveal hidden C&C server domains or IPs quickly.

### 4.3 Validating Our Techniques

R2D2 is implemented in C++ and Python leveraging S2E [46] for concolic analysis. Before our large-scale measurement, we needed to verify that R2D2 can identify dead drop resolvers and decoders correctly.

Table 4.3: Validating Dead Drop Domain Candidate Identification and Confirmation.

| Malware      | #Variants | DDR Domain                    | TP FP FN | Domain Origin  | TP FP FN | DDR Confirmation | TP FP FN | Time (s) |
|--------------|-----------|-------------------------------|----------|----------------|----------|------------------|----------|----------|
| razy         | 5         | www.twitter.com               | 5 0 0    | hard-coded     | 5 0 0    | tag propagated   | 5 0 0    | 148      |
| doina        | 5         | drive.google.com              | 5 0 0    | hard-coded     | 5 0 0    | implicit flow    | 5 0 0    | 230      |
| kryptik      | 5         | www.pastebin.com              | 5 0 0    | hard-coded     | 5 0 0    | tag propagated   | 5 0 0    | 484      |
| midie        | 5         | bitbucket.org                 | 5 0 0    | hard-coded     | 5 0 0    | tag propagated   | 0 5 0    | 130      |
| cryptolocker | 5         | sftnwbyrkswt.net <sup>1</sup> | 5 0 0    | DGA            | 3 0 2    | N/A              | 0 0 0    | 887      |
| zusy         | 5         | www.pastebin.com              | 5 0 0    | hard-coded     | 5 0 0    | tag propagated   | 5 0 0    | 332      |
| comnie       | 5         | www.github.com                | 5 0 0    | hard-coded     | 5 0 0    | tag propagated   | 5 0 0    | 417      |
| <b>Total</b> | 35        | Accuracy (100%)               | 35 0 0   | Accuracy (96%) | 33 0 2   | Accuracy (90%)   | 25 5 0   | 375      |

1: This is one of many DGA domains that R2D2 identified.

### 4.3.1 Dead Drop Resolver Identification and Confirmation

Table 4.3 presents R2D2’s evaluation to identify domains and DDR integration in malware. To do so, we use 35 Windows malware samples from our dataset and manually reverse engineered them to determine the ground truth metrics. Columns 1-2 list malware families and variants. The remaining columns display the ground truth domains identified, their origin, and how the DDR capabilities were confirmed, including accuracy metrics — true positive (TP), false positive (FP), and false negative (FN). Since we seek to identify hybrid DDR+DGA malware, we used a known DGA-based malware, *cryptolocker* (Row 5) to ensure DGA detection. This ensures that we test all of R2D2’s design components.

R2D2 correctly identified all domains (35 TPs) and achieved a 96% accuracy in identifying 33 of 35 domain origins. R2D2 also achieved 90% accuracy confirming 25 of 30 DDR-based malware.

Upon closer inspection, we found FNs in *cryptolocker* occurred because 2 variants contained a hard-coded domain that the malware attempted to connect to first. If it failed, then the DGA was used. R2D2 rightly classified the hard-coded domain but missed the DGA component in 2 of 5 *cryptolocker* malware. However, our investigation confirmed that these are rare occurrences.

The 5 FPs occurred in the *midie* sample because of over tainting. When data was read from `bitbucket.org`, R2D2 allowed the malware to continue to read data 128 bytes at a time until a hard-coded value specifying the amount of data to read was reached. However, at the last iteration, 128 bytes was an over-approximation whereby R2D2 symbolized a portion of the buffer more than the prescribed read limit. This resulted in symbolizing a parameter used in the subsequent malicious outbound connection. Further investigation confirmed that `bitbucket.org` was accessed by the malware and data was retrieved, but the data was used in the `WriteFile` API before the newly written file was used in `ShellExecute`. This confirms that `bitbucket.org` was used to host a dropped file but not as a DDR.



Given the low number of FPs (5) and FNs (2) and the high accuracy of 100%, 96%, and 90% for domain identification, domain origin, and DDR confirmation, respectively, R2D2 is ready for large-scale deployment. Moreover, averaging a run time of 375 seconds, R2D2 can quickly report details of DDR integration toward rapid remediation by authorities.

#### 4.3.2 Decoding Algorithm Comparison

Before we identify decoders in malware, we must first answer: Given two different implementations of the same algorithm, will R2D2 still be able to identify that algorithm? Put simply, will R2D2's reference decoders (from Table 4.2) work regardless of a malware's implementation? Answering this requires: (1) comparing source code similarity to ensure all decoder implementations are different and (2) comparing symbolic expression equivalence of all pairs of decoders to show that irrespective of implementation, only decoders of the same class (e.g., all Base64 decoders) match with high confidence.

##### ***Source Code Similarity***

We choose up to 3 implementations of each algorithm from Table 4.2 that are available on open-source software repositories. We use Moss [161], a system for detecting software similarity that is widely used in academia to detect code plagiarism. If we can show that Algorithm 2 matches decoders of the same class even though their implementations differ, then R2D2's approach will work irrespective of a malware's decoder implementation. As shown in Table 4.5, no differing algorithms (600 comparisons) overlap, as formalized in Row 6, where  $|R|$  is the length of the set containing all decoder algorithm implementations and  $m$  and  $n$  are indices of different algorithms in the set. The only matches occurred when one algorithm was compared with itself (25 comparisons), which is expected. This verifies that each of the decoding algorithms selected for R2D2 is a different implementation.

Table 4.4: Baseline Comparison for Decoding Algorithm Similarity via Symbolic Expressions Matching.

|          |    | Base16 |     |     | Base32 |     |     | Base64 |     |     | Base85 |     |     | XOR |     |     | Chr Sub. |     |     | Str $\leftrightarrow$ Int |     | Rotate |     |     | Str Prs |     |
|----------|----|--------|-----|-----|--------|-----|-----|--------|-----|-----|--------|-----|-----|-----|-----|-----|----------|-----|-----|---------------------------|-----|--------|-----|-----|---------|-----|
|          |    | v1     | v2  | v3  | v1     | v2  | v3  | v1     | v2  | v3  | v1     | v2  | v3  | v1  | v2  | v3  | v1       | v2  | v3  | S→I                       | S←  | v1     | v2  | v3  | v1      | v2  |
| Base16   | v1 | 100    | 98  | 92  | 16     | 19  | 11  | 28     | 0   | 16  | 21     | 9   | 42  | 0   | 0   | 0   | 0        | 0   | 0   | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
|          | v2 | 89     | 100 | 72  | 15     | 8   | 23  | 50     | 51  | 52  | 32     | 14  | 7   | 9   | 8   | 8   | 0        | 0   | 0   | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
|          | v3 | 96     | 91  | 100 | 14     | 23  | 22  | 12     | 9   | 27  | 11     | 24  | 22  | 0   | 0   | 0   | 0        | 0   | 0   | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
| Base32   | v1 | 12     | 11  | 14  | 100    | 97  | 98  | 62     | 68  | 64  | 1      | 22  | 13  | 0   | 0   | 0   | 0        | 0   | 0   | 7                         | 1   | 0      | 0   | 0   | 0       | 0   |
|          | v2 | 1      | 1   | 13  | 98     | 100 | 97  | 2      | 12  | 9   | 0      | 17  | 8   | 0   | 0   | 0   | 0        | 0   | 0   | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
|          | v3 | 13     | 21  | 16  | 92     | 83  | 100 | 3      | 9   | 22  | 13     | 4   | 3   | 0   | 0   | 0   | 0        | 0   | 0   | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
| Base64   | v1 | 70     | 14  | 2   | 0      | 0   | 27  | 100    | 44  | 77  | 19     | 0   | 0   | 0   | 0   | 0   | 0        | 0   | 0   | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
|          | v2 | 1      | 16  | 4   | 3      | 4   | 4   | 97     | 100 | 100 | 21     | 14  | 5   | 0   | 0   | 0   | 0        | 0   | 0   | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
|          | v3 | 4      | 40  | 16  | 11     | 3   | 21  | 89     | 100 | 100 | 7      | 18  | 10  | 0   | 0   | 0   | 0        | 0   | 0   | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
| Base85   | v1 | 17     | 45  | 46  | 23     | 42  | 62  | 41     | 37  | 19  | 100    | 91  | 93  | 0   | 0   | 0   | 0        | 0   | 0   | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
|          | v2 | 10     | 61  | 42  | 29     | 37  | 58  | 38     | 52  | 43  | 87     | 100 | 92  | 0   | 0   | 0   | 0        | 0   | 0   | 0                         | 1   | 0      | 0   | 0   | 0       | 0   |
|          | v3 | 32     | 15  | 29  | 22     | 12  | 33  | 19     | 27  | 20  | 89     | 92  | 100 | 0   | 0   | 0   | 0        | 0   | 0   | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
| XOR      | v1 | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 100 | 100 | 100 | 11       | 0   | 0   | 11                        | 0   | 0      | 0   | 0   | 0       | 0   |
|          | v2 | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 100 | 100 | 100 | 0        | 0   | 0   | 11                        | 0   | 0      | 0   | 0   | 0       | 0   |
|          | v3 | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 100 | 100 | 100 | 0        | 0   | 0   | 11                        | 0   | 0      | 0   | 0   | 0       | 0   |
| Char Sub | v1 | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0   | 0   | 0   | 100      | 92  | 94  | 0                         | 0   | 0      | 0   | 0   | 0       | 0   |
|          | v2 | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0   | 0   | 0   | 86       | 100 | 84  | 11                        | 0   | 0      | 0   | 0   | 0       | 0   |
|          | v3 | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0   | 0   | 0   | 92       | 91  | 100 | 11                        | 0   | 0      | 0   | 0   | 0       | 0   |
| Str, Int | →  | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 2   | 4   | 11  | 11  | 11  | 1        | 0   | 0   | 100                       | 4   | 0      | 0   | 0   | 0       | 0   |
|          | ↑  | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0   | 0   | 0   | 0        | 0   | 0   | 22                        | 100 | 0      | 0   | 0   | 0       | 0   |
| Rotate   | v1 | 6      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 6   | 4   | 6   | 2        | 6   | 0   | 0                         | 0   | 100    | 100 | 100 | 0       | 0   |
|          | v2 | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0   | 0   | 0   | 0        | 0   | 0   | 1                         | 0   | 100    | 100 | 100 | 0       | 0   |
|          | v3 | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0   | 0   | 0   | 0        | 0   | 0   | 1                         | 0   | 100    | 100 | 100 | 0       | 0   |
| Str Prs  | v1 | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0   | 0   | 0   | 1        | 0   | 0   | 11                        | 0   | 13     | 3   | 3   | 100     | 99  |
|          | v2 | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0      | 0   | 0   | 0   | 0   | 0   | 2        | 0   | 0   | 11                        | 0   | 19     | 2   | 3   | 99      | 100 |

Table 4.5: Baseline Comparison for Decoding Algorithm C/C++ Source Code via Moss.

|    | <b>Base16</b> |     |     | <b>Base32</b> |     |     | <b>Base64</b> |     |     | <b>Base85</b> |     |     | <b>XOR</b> |     |     | <b>Char Sub.</b> |     |     | <b>Str <math>\leftrightarrow</math> Int</b> |     | <b>Rotate</b> |     |     | <b>Str Prs</b> |     |
|----|---------------|-----|-----|---------------|-----|-----|---------------|-----|-----|---------------|-----|-----|------------|-----|-----|------------------|-----|-----|---|-----|---------------|-----|-----|----------------|-----|
|    | v1            | v2  | v3  | v1            | v2  | v3  | v1            | v2  | v3  | v1            | v2  | v3  | v1         | v2  | v3  | v1               | v2  | v3  | S→I   | S←  | v1            | v2  | v3  | v1             | v2  |
| v1 | 100           | 0   | 0   | 100           | 0   | 0   | 100           | 0   | 0   | 100           | 0   | 0   | 100        | 0   | 0   | 100              | 0   | 0   | 100   | 0   | 100           | 0   | 0   | 100            | 0   |
| v2 | 0             | 100 | 0   | 0             | 100 | 0   | 0             | 100 | 0   | 0             | 100 | 0   | 0          | 100 | 0   | 0                | 100 | 0   |   |     | 0             | 100 | 0   |                |     |
| v3 | 0             | 0   | 100 | 0             | 0   | 100 | 0             | 0   | 100 | 0             | 0   | 100 | 0          | 0   | 100 | 0                | 0   | 100 | 0   | 100 | 0             | 0   | 100 | 0              | 100 |

$\forall m, n \in [0, |R|] \wedge m \neq n : Moss(R_n, R_m) = 0\%$

## Symbolic Expression Matching

Table 4.6: Baseline Comparison for Decoding Algorithm Similarity via Symbolic Expressions.

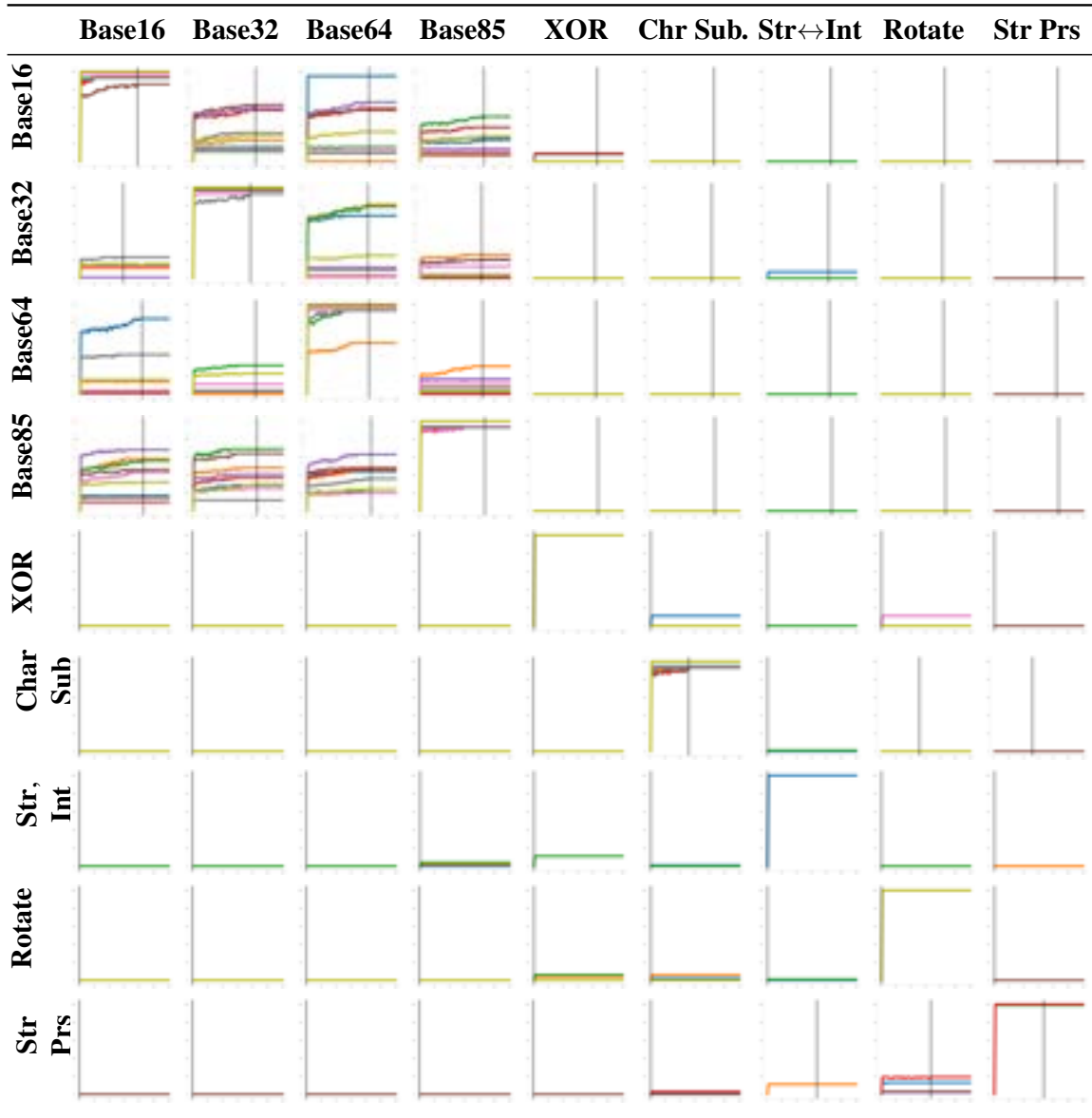


Table 4.4 presents the evaluation of Algorithm 2 which takes two symbolic expressions as input. The comparison of a pair of expressions is not limited to one evaluation. Since we specify a symbolic input size of 8 bytes, the decoders’ expressions could concretize to  $256^8$ , or 4.2 billion, possible values. Ideally, we would like to evaluate the entire input space, but that is prohibitively time-consuming, especially when considering 625 comparisons. So,

we execute each comparison for 2 hours. We find this to be sufficient because the number of `nomatch` versus `match` (Algorithm 2) results plateau and stabilize for more than 30 minutes within 2 hours, convincing us of the overall percentage match of both algorithms being considered. In the worse case, when Base16 is compared with Base85 (Table 4.6, Column 4, Row 1), the last plateau begins at 85 minutes and continues until the 2-hour mark. We present a view of the plateau for each set of comparisons in Table 4.6.

Interestingly, we found that some decoder algorithms cannot derive the same output on corner-case inputs. In fact, we observed this in practice when testing decoder algorithms concretely. These corner cases are expected in algorithms developed differently as decoding functions are not rigorously tested to ensure completeness or include different error handling. Still, these differences are negligible compared to the difference with other algorithm classes.

As Table 4.4 shows, each cell represents the percentage of expression matching. The majority (474/554<sup>6</sup> or 85.5%) of comparisons of algorithms from different classes result in 0% match, which is expected. However, notice Base64 v1 compared with the other Base64 versions (Row 7). We expect them to match at a high rate, even amid mismatching corner cases. Our investigation revealed that the 44% and 77% comparison percentages for v2 and v3 are due to error checking in v1. As a reminder, all decoder implementations take 8 bytes as input. In fact, v1 has error checking to ensure the input size is  $> 4$ . When R2D2 forks at this error check, it will take two paths: (1) successful return and (2) failure where null is returned. As discussed previously, the symbolic data is constrained for the second algorithm in the comparison based on the exploration of the first comparison. So, where v1 had 1 failure and 1 success in the above example since v2 has less strict error checking, it succeeded in both, accounting for their low matching percentage. Conversely, when Base64 v2 and v3 (Rows 8 and 9) are compared with v1 (Column 7), they match at 97%, 89%, and 100%, since the former does not have excessive error checking. Since all

---

<sup>6</sup>For the other 71 of 625 comparisons, decoders are compared with themselves.

implementations take 8 bytes as input, when v2 is explored first and succeeds, the resulting constrained symbolic expression used in v1 ensures that R2D2 takes the success path in v1, resulting in a high percentage match.

Table 4.4 also shows an interesting trend within the Base decoding classes (Base 16, Base 32, etc.). We are not surprised that 12/144 Base class comparisons (all Base versions compared with themselves) match at 100% (top left to bottom right diagonal of the Base class comparisons). However, of the remainder, we only observe 6/144 with a 0% match when we would expect to find 108 with a 0% match (when one Base class is compared with another). Base algorithms depend on a table of data to index for character translation. As a result, all Base algorithms match to a certain extent based on the indexing of tables, but not enough to deduce a confident equivalence. There are other overlapping instances of algorithms from differing classes, but they are also too low to be considered a match.

From each class, we select the best performing implementation, meaning that they match >90% within their class and <25% across the others (highlighted in orange on the left of Table 4.4). These are built into R2D2 as “reference implementations.” As subsection 4.2.3 described, new algorithms can always be added to R2D2. These results show that R2D2 can identify decoders with different implementations in malware.

### 4.3.3 Decoding Algorithm Identification

We now must verify that R2D2 can identify decoders in malware using Algorithm 3. We use the same malware ground truth set from subsection 4.3.1. Since both *midie* and *crytolocker* (Table 4.3) are not DDR-based malware, we omit them from decoder evaluation. Table 4.7 presents our validation of R2D2’s decoder identification in 25 manually reverse engineered malware. Columns 1-2 list the malware family and variants. Column 3 lists the decoders. The remaining columns list the accuracy metrics. Overall, R2D2 achieves a 94% accuracy in correctly identifying 45 of 50 decoders.

We notice that FPs occur in *doina* malware. We expect that DDR-based malware en-

Table 4.7: Validating Decoder Identification.

| Malware      | #Variants | Decoder        | Metrics |    |    |
|--------------|-----------|----------------|---------|----|----|
|              |           |                | TP      | FP | FN |
| razy         | 5         | Base64         | 5       | 0  | 0  |
|              |           | String Parsing | 5       | 0  | 0  |
| doina        | 5         | String Parsing | 0       | 5  | 0  |
| kryptik      | 5         | Base64         | 5       | 0  | 0  |
|              |           | String Parsing | 5       | 0  | 0  |
| zusy         | 5         | XOR            | 5       | 0  | 0  |
|              |           | Base16         | 5       | 0  | 0  |
| comnie       | 5         | Base64         | 5       | 0  | 0  |
|              |           | XOR            | 5       | 0  | 0  |
|              |           | Char Rotate    | 5       | 0  | 0  |
| <b>Total</b> | 25        | Accuracy (94%) | 45      | 5  | 0  |

codes the C&C rendezvous points, but in this case, *doina* requests a text file from Google Drive containing an IP address in plaintext. We would expect R2D2 not to identify any decoders, but it identified String Parsing (Row 5). This occurs because the malware checks for the IP address format using string search routines, similar to what is needed for string parsing. R2D2 may be prone to FPs when dealing with plaintext data, but the effect is negligible since authorities will immediately have the C&C rendezvous point.

For all other malware families and variants, R2D2 detected the IO boundary and successfully compared sub-expressions for decoder matching. Given the low number of FPs (5) and no FNs, and 94% accuracy, R2D2 provides the means to identify decoders in malware effectively.

#### 4.4 Dead Drop Resolver Findings

Deploying R2D2 revealed 10,170 DDR-based malware in our dataset. Figure 4.3 illustrates the relative prevalence of DDR use since 2017. The red line represents the quantity of malware from a specific year. The orange line represents the quantity of DDR-based malware

that R2D2 identified. The normalized result, or blue line, indicates the trend of DDR use over the last 5 years. Note that there has been a notable increase in the relative amount of DDR-based malware in the wild over the last 2 years. This is not surprising given the benefits that DDRs offer malware authors.

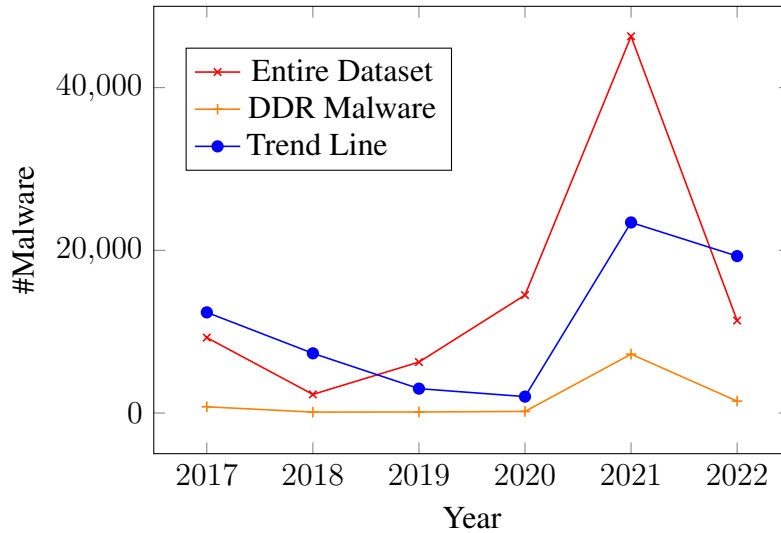


Figure 4.3: DDR Malware Trends Since 2017.

Since we designed R2D2 to study DDR-based malware, it will not reveal how the additional malware in our dataset work. However, a cursory investigation revealed that some of the remaining samples also rely on web apps, but not as a DDR. We leave the exploration of other web app abuse in malware for future work. The remainder used HTTP-based communication with non-web app domains. This is in line with what has been observed in prior works that reported up to 70% of malware use HTTP-based communication [109, 21] to connect to malicious domains.

#### 4.4.1 Dead Drop Resolver-Based Malware Discoveries

We deployed R2D2 to analyze 100k malware and present the results in Table 4.8. Column 1 lists the 15 web apps identified. Columns 2-3 list the number of malware and families. Column *Domain Origin* lists the number of malware containing hard-coded versus DGA domains, and Column *CTP* (concolic taint propagation) shows how many samples relied



Table 4.8: Distribution of Web App Domains used for DDR-based Malware Across our Dataset.

| Web App Domains       | #Malware            | #Families | Domain Origin       |     | CTP <sup>1</sup>    |                  |
|-----------------------|---------------------|-----------|---------------------|-----|---------------------|------------------|
|                       |                     |           | Hard-Coded          | DGA | Propagated          | Implicit Flow    |
| pastebin.com          | 6,053               | 30        | 6,053               | 0   | 5,712               | 341              |
| blockchain.info       | 1,888               | 59        | 1,888               | 0   | 1,765               | 123              |
| blockcypher.com       | 1,437               | 41        | 1437                | 0   | 1,398               | 39               |
| bitaps.com            | 722                 | 16        | 722                 | 0   | 707                 | 15               |
| docs.google.com       | 616                 | 10        | 616                 | 0   | 571                 | 45               |
| blockr.io             | 200                 | 5         | 200                 | 0   | 189                 | 11               |
| dropbox.com           | 204                 | 6         | 204                 | 0   | 189                 | 15               |
| googleusercontent.com | 151                 | 9         | 151                 | 0   | 141                 | 10               |
| coinmarketcap.com     | 50                  | 10        | 50                  | 0   | 47                  | 3                |
| twitter.com           | 34                  | 8         | 22                  | 12  | 34                  | 0                |
| blockchain.com        | 5                   | 7         | 5                   | 0   | 5                   | 0                |
| github.com            | 7                   | 3         | 7                   | 0   | 7                   | 0                |
| blockstream.info      | 3                   | 3         | 3                   | 0   | 3                   | 0                |
| drive.google.com      | 3                   | 2         | 3                   | 0   | 3                   | 0                |
| wordpress.com         | 4                   | 2         | 4                   | 0   | 4                   | 0                |
| <b>Total</b>          | 11,377 <sup>2</sup> | 154       | 11,365 <sup>2</sup> | 12  | 10,775 <sup>2</sup> | 602 <sup>2</sup> |

1: Concolic Taint Propagation.

2: This is not the total number. Most malware using cryptocurrency web apps use more than 1 as backup.

on explicit flow versus implicit flows. R2D2 identified 10,170 DDR-based malware across 154 families that abuse 15 web apps and 275 unique accounts. R2D2 also found that 1,054 (10.3%) DDR-based malware use multiple web apps for backup.

The Pastebin web app accounts for the most DDR-based malware (Row 1) at 6,053 ( $\approx 59\%$ ). This is not surprising, as Pastebin has long been used for malicious purposes [162]. However, it has generally been used to host stolen content or malware to be dropped on victim systems. This work is among the first to expose its pervasiveness as a hosting platform for hidden C&C rendezvous points.

Blockchain.info is the next most prevalent web app accounting for 1,888 samples ( $\approx 19\%$ ). In fact, 3/4 highest-ranking web apps are crypto web apps. There are 7 crypto web apps that enable DDR-based malware (Rows 2-4, 6, 9, 11, and 13), totaling 3,098 DDR-based malware. Notably, 3,098 malware in our dataset use a crypto web app. Of those, 1,054 use

one or more domains for backups. However, even with backups, each malware uses one wallet ID to retrieve recent transactions, which are decoded into a C&C server IP address. We present the list of the 75 identified wallet IDs in Table 4.12, Table 4.13, and Table 4.11.

R2D2 also identified popular web apps, including Twitter, Google, Github, and Dropbox. However, they account for fewer occurrences than expected, totaling 1,019 or  $\approx 10\%$  of all DDR-based malware. Several works have studied popular web app abuse though none considered DDR, so malware authors are likely using less popular web apps to reduce suspicion.

Next, we observe that *miniduke* uses a DGA to generate Twitter accounts (Column *Domain Origin*). This hybrid DDR+DGA malware is unique given that only 12 have been found in our study, accounting for 0.12% of all DDR-based malware. We conclude that although the complexity of this malware can pose challenges for authorities, the plethora of works that counteract DGA-based malware may motivate malware authors to use DDRs solely.

Next, we also notice the distribution of malware where the standard concolic taint propagation proved sufficient (Column *CTP*). In 10,775 instances of web app abuse in malware, the data read from the web app was propagated via data dependence. Yet, only 602 instances of tag propagation crossed implicit flow blocks that broke propagation before being remedied by R2D2 (subsubsection 4.2.2). This occurs when the buffer that holds the decoding string receives its data from hard-coded malware values dependent upon a previous condition (e.g., if byte 1 of the encoded string is 'a', then store '/' in the decoded string buffer). However, these are rare occurrences.

In total, we find that 10,170 of the 100k malware are DDR-based malware illustrating the prevalence of this practice. In fact, Netskope reported that more than 66% of malware downloads come from web apps [81]. However, what was unknown is how many use web apps for DDR, which R2D2 finds to be  $\approx 10\%$ .

#### 4.4.2 Decoders Identified

Table 4.9: The Number of Occurrences of Decoders in the DDR-based Malware.

| Decoder               | #Malware | #Families |
|-----------------------|----------|-----------|
| String Parsing        | 7,424    | 92        |
| Base64                | 6,305    | 107       |
| Exclusive OR          | 4,474    | 61        |
| Base16                | 3,166    | 60        |
| String to Integer     | 2,854    | 23        |
| Integer to String     | 2,854    | 23        |
| Character Rotation    | 2,135    | 29        |
| Character Subtraction | 711      | 11        |
| Base32                | 0        | 0         |
| Base85                | 0        | 0         |

Table 4.9 presents R2D2’s decoder identification in 10,170 DDR-based malware. The most common decoder is String Parsing, occurring 7,424 DDR-based malware. Since over 78% DDR-based malware use multiple encoding techniques to impede analysis, they must parse the data received from the web app to apply decoding properly. R2D2 provides the ability to detect this so authorities can directly decode these more complex obfuscation schemes. Base64 is the next most popular decoder occurring in 6,305 (62%) DDR-based malware. Base64 affords the benefits of ensuring data goes unmodified during transport. Thus, DDR-based malware commonly use it.

Surprisingly, R2D2 identified  $\approx 6\%$  of DDR-based malware with plaintext C&C rendezvous points. For example, a Pastebin DDR-based malware retrieved `81.30.144.81:39431`. As discussed in subsection 4.3.3, R2D2 often reports String Parsing in malware with plaintext rendezvous points. In this case, the malware parsed `:` as the delimiter to separate the IP address from the port number.

To our surprise, the most popular decoders are not complex. In fact, if the correct encoding is identified, decoding becomes a trivial task. Yet, malware authors continue to use them because it is difficult to identify the encoding type, especially when multiple are used. In total, R2D2 identified 29,923 decoders used in 10,170 DDR-based malware

## Not Found (#404)

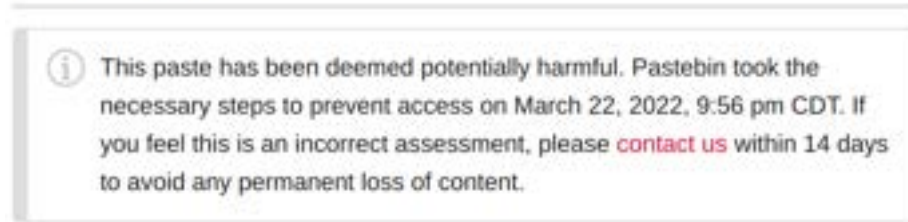


Figure 4.4: A Pastebin Account Removed.

for an average of 2.25 decoders per malware. R2D2 provides authorities with a *recipe* of decoders enabling the rapid identification of malicious C&C server rendezvous points toward counteraction.

### 4.4.3 Towards Remediation

R2D2 revealed the requisite evidence to pursue botnet counteraction: the contacted web app, confirmation of its abuse to host a C&C rendezvous point, and the decoding recipe to reveal the rendezvous point. With this information, authorities can submit transactions to malicious crypto wallets to sinkhole the botnet, as demonstrated in [163] or work with web app providers to replace encoded C&C rendezvous points with their own. Furthermore, authorities can use information from R2D2 to monitor the botnet even when it migrates the C&C server. As recently demonstrated, botnet monitoring is crucial to enabling successful disruption and takedown attempts [21]. In this research, since we lack authority to pursue active counteraction, we sought cooperation from web app providers.

**R2D2: Real-World Impact.** R2D2 found 275 web app accounts, and we reported them to the web app providers. Thus far, Pastebin, WordPress, and BitcoinAbuse confirmed our findings and took action against 9,155 DDRs (90% of our total findings). Specifically, Pastebin disabled (Figure 4.4) the offending accounts we discovered directly resulting in 6,053 infected victim systems unable to communicate with the malicious web app account (60% of all DDR-based malware that R2D2 identified) and any newly infected systems cannot access the C&C server. Similarly, WordPress also responded to our findings and

**From:** Automattic Trust & Safety <abuse@wordpress.com>  
**Sent:** Tuesday, March 29, 2022 8:22 PM  
**To:** <redacted>  
**Subject:** [-] Re: abuse report

---

**Automattic Trust & Safety (Automattic)**  
Mar 30, 2022, 0:22 UTC

Hello,

Thank you very much for your report.

The sites in question have been removed from  
WordPress.com for violating our Terms of Service.

Automattic Trust & Safety

Figure 4.5: Response From WordPress.

also removed the offending accounts from their platform (Figure 4.5). Although WordPress DDR-based malware only accounted for 0.04% of all 10,170 DDR-based malware discovered, it illustrates the range of options that malware authors have for DDR botnets.

For the 3,098 DDR-based malware (30%) that use crypto wallets to retrieve C&C rendezvous points, they must connect to one of many blockchain explorer web apps (e.g., `blockchain.info`) to search/read the blockchain. Disk space requirements make downloading the entire blockchain to the victim system impractical. Thus, although blockchains are immutable, these web apps control access to view crypto transactions. A practical solution towards remediation is to flag wallet IDs used in cybercrime and block viewing access to prevent malware from retrieving blockchain information. To this end, we submitted the 75 wallets IDs, listed in Table 4.12, Table 4.13, and Table 4.11 that R2D2 discovered to BitcoinAbuse [164] to publicly document that they have been used in malware. However, since R2D2 reveals how the crypto transactions are decoded to C&C server IP addresses, authorities can submit transactions to the offending accounts to sinkhole the botnet [163].

Lastly, Twitter negatively responded to our findings and did not remove the malicious account since it “hasn’t broken our safety policies”, as illustrated by their response in Fig-

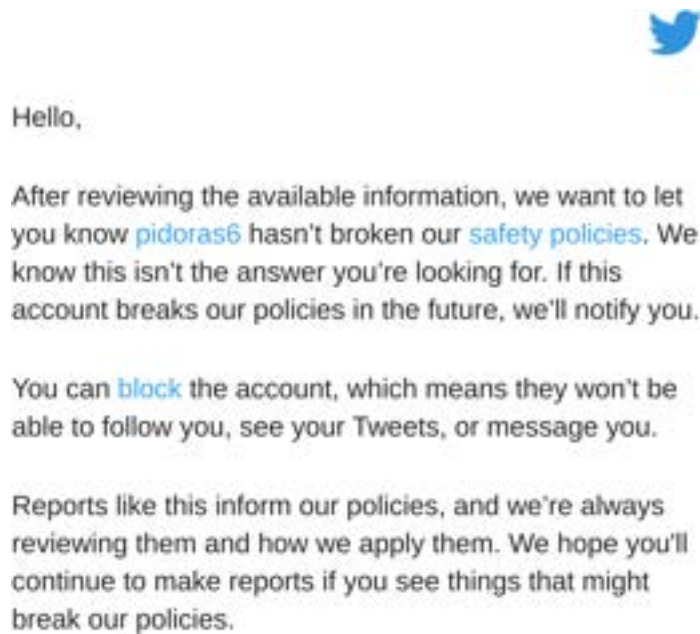


Figure 4.6: Response From Twitter.

ure 4.6. Of the remainder, the accounts for 774 malware were already taken down at the time of our study. We are awaiting a response from other web app providers hosting the other 241 DDRs.

#### 4.4.4 Packed Dead Drop Resolver-Based Malware

Table 4.10: Packed Dead Drop Resolver-Based Malware.

| Packer Type [92] | Packer [115]        | #Malware     |
|------------------|---------------------|--------------|
| Type-I           | BobSoft Mini Delphi | 845          |
|                  | UPX                 | 234          |
|                  | FASM                | 1            |
| Type-II          | PeCompact           | 145          |
| Type-III         | ASPack              | 2            |
|                  | ASProtect           | 2            |
| Type-IV          | tElock              | 1            |
| <b>Total</b>     |                     | <b>1,230</b> |

Considering malware obfuscation via encoding led us to consider packing techniques used on the binaries themselves. Using PackerID [115], we present Table 4.10 which lists

the packer type in Column 1, as defined by Ugarte-Pedrero et al. [92] and packers in Column 2. Column 3 shows the number of malware. Only 12% (1,230) of the DDR-based malware are packed. Furthermore, 1,080 (89%) of packed samples use Type-I packers which are easier to unpack. While Type-II packers use added layers of packing, they are still not difficult to unpack. We only identified 5 instances (0.4%) of Type-III and IV packers.

Given the prevalence of the decoders identified in 9,559 (94%) of the 10,170 DDR samples, we assume malware authors are not devoting resources to packing in hopes that obfuscation and DDR integration will prove sufficient for robust botnets. We hope our study of DDR-based malware will shed light on this under-explored but trending practice and enable authorities to take action against DDR-enabled botnets.

Table 4.11: Bitcoin Wallet IDs (1/3).

| Web App         | Wallet ID                          |
|-----------------|------------------------------------|
| blockcypher.com | 33fde6e00a62995ddd4977b5cf7b8bc55c |
|                 | 1VocauiabutZLvzBau7V6QgCC7WQnmU1n2 |
|                 | 1BjVeaZBMA9QEweeRfK6nftzDPbr7jMaDk |
|                 | 161fPjdCt5H9uYawPpPT4poc8RBcLFaE3R |
|                 | 1CeLgFDu917tgtunhJZ6BA2YdR559Boy9Y |
| bitaps.com      | 17gd1msp5FnMcEMF1MitTNSsYs7w7AQyCt |
|                 | 1CMRScsrPe2N4HwPpNKcHUhfCJXUm2Cx6  |
|                 | 1HTDy9SkfhwaNCXFA8wFCvN53f3iGpm8kb |

## 4.5 Discussion

### 4.5.1 Adversarial Response

A malware author may use cryptographic techniques to encrypt the C&C rendezvous point and thwart R2D2. However, there are several tools/techniques [165, 166, 167] available that can be integrated into R2D2 towards decryption. It is also possible for malware authors to develop complex functions that impede a tractable computation for our symbolic solver

Table 4.12: Bitcoin Wallet IDs (2/3).

| Web App           | Wallet ID  |
|-------------------|--|
| coinmarketcap.com | 1BkeGqpo8M5KNVYXW3obmQt1R58zXAqLBQ<br>1N94rYBBCZSnLoK56omRkAPRFrpr5t8C1y   |
| blockstream.info  | 1qre9cdrqdagy0p2sww2dvp7td86kws09v<br>1qj2h87z0v8u7ddp823apvjzpu5asssfpy   |
| blockchain.info   | 1Lud76Q98VRHCUiyK7XUs7AgFofrqXep78<br>15GqSWnxEFZezUCcGjhBMknA1PB7aYNXC1<br>18sHYU49vUFk6TN6G2Pj6DSCUzkbLvwJtc<br>1DsyxmgvkbTnLBnCXWyyNaDNhgmzib4mp<br>1BkeGqpo8M5KNVYXW3obmQt1R58zXAqLBQ<br>13LHbsf1CWgat1ZLYYomsjeeybvCD7ZUxh<br>3ab5ab9511cf52565314425424d0b0b978<br>1qtkmks24vyuemjm6w3j3qagyn2pu3d93y<br>1NL67bQ8dPbfxLKcXBpuE3n8H5AsExBvwt<br>38D2P6apsGhghkGK4mSAMB9yr5enXW6iUy<br>32Lsw4r5YGLS5qhZsgp1b2kk1xTbf7T4Wf<br>16nA62oxxsDgc2R2NoW6WtFrZkB3XLvVpb<br>323c2a4e57b5ba21687fe7ce5918ceaf4a<br>1Dhf71bPe3wQ4At9YSaEVXGyhwzFiKNdBo<br>1q97764a4dnuzfd5dxxyhgggyn7de9z978<br>1qsce320qf73s9v593p0jxfs46q7nh0zus<br>1NxsR82Efaqbnt3c9QQUoYJpejwFtDrnNe<br>1Eyx9PKb1bs9X7n4UK7JHnzSxedyUvHimE<br>1FK6Y4BcHV5jQ6nPJTL83EyrhLzWGTvkfv<br>1PwAEK6Zp371Vegwp38XzP4nULikzUrCra<br>1K4GnSGtoH7qx4SvpoJ3v3Nv2yZg6v5sDY<br>1EYnNjRKWqFBRLe53Ui1HVwxDwK1gqsKGp<br>3Gf7NRXDKtAzeTYG9fwHLg9snpSi2AZpyj<br>3QKjFKdqtJi34UnFwdA8VsaV2NkRgmaUnf<br>3NXdpbSZqe3sniegAgzEpo7YisDqVQiL6y<br>1KVwMFZw4QUuoqdeWtehDeB7qLhx3ncGVV<br>1KQheJU4ZwXvMdoLevVhDVVHU1zWUMcDp<br>1qwqr922tvn69gsdhn7fcayspwt202fzer<br>33u5t5A8qvQFdwVMANDFS91LGMw68fiaMc<br>1GAEMH9wiX8LqJm7v8oKLdgm5Zr3msE6E3 |
| blockr.io         | 1cptcvckjajnkdd7psapv3cakupvd4mcmt<br>17gd1msp5FnMcEMF1MitTNSsYs7w7AQyCt<br>1HTDy9SkfhwaNCXFA8wFCvN53f3iGpm8kb<br>1a4778fd2ba2b8ae98c573defa3b0e86d8<br>1GcnsLs7C31uuroNmUHwwbB5xQeNvm63Ee   |



Table 4.13: Bitcoin Wallet IDs (3/3).

| Web App   | Wallet ID  |   |
|-----------|--|---|
| blockr.io | 1CpTCVckja jNKDd7PsApV3cAkunVd4Mcmt<br>1d6037414ac2bbf101eadae4d4c4d57e98<br>1CMRScsrPe2N4HwPpNKcHUhfCJXUm2Cx6<br>17bf8ba6d1bb9e5f03b0946d467fca7887<br>1b354ee81d0ea177be5355b8a430db1b22<br>3a0ed3db93838620fee7aed8c87f3ebacb<br>11486a040f0cf7511a53f7610958f2a109<br>35ef6c71fafa9e30ee56d312dd626999ac<br>1e77054da43c04f19b628a7ea5bfc6d1ca<br>1e52ba07c17cc49995c915209b23b23ad6 |   |
|           | blockcypher.com  | 1HTDy9SkfhwaNCXFA8wFCvN53f3iGpm8kb<br>1016d7ceff188e9fe32e68e9761bd811f3<br>17gd1msp5FnMcEMF1MitTNSsYs7w7AQyCt<br>1BkeGqpo8M5KNVYXW3obmQt1R58zXAqLBQ<br>3916a96ba7bfc95ed103aff4286360e820<br>1ML94w1SCudkiFHaEwYqTmKGTkywvVBuZg<br>1a4778fd2ba2b8ae98c573defa3b0e86d8<br>1GcnsLs7C31uuroNmUHwwbB5xQeNvm63Ee<br>193896af781481f195a4c55cbf053b7e95<br>1CMRScsrPe2N4HwPpNKcHUhfCJXUm2Cx6<br>3bcec9103e14bd8969f2d1f2e14bd72399<br>3e3c52d8aeed29d2e5f2835061f01fa758<br>18ecb27aff6d1c6a889f810c50eb72e565<br>1d8c213480d883fc2c4a001ecfb106f241<br>1CpTCVckja jNKDd7PsApV3cAkunVd4Mcmt<br>198009d287c818d2a9aa72f7f828c19c84<br>19hi8BJ7HxKK45aLVdMbze6oTSW5mGYC82<br>1N9ALZUgqYzFQGDXvMY5j1c7PGMMGYqUde<br>133be6e6ccca5bb6b3e3aaa34cf14a374a<br>34d153ae12ebfe18cea39ddc07d514865b<br>14bbtRSruiXHtvofYgB24Wdpma1Bx6RSof<br>19ZN4JM9ZH2nLc3PZh85n3t1WVz jBKD39D<br>14a0b3c26dc368d1b69862eb28fd8648fd<br>1ALuqPer2DSD9YyU9nrZz6NR1dDwCQLnE7<br>1BYZgQnu3M86ra95Jywj5xiL2fE7Nbn64q<br>1Fbhv84haM4TiwcR71WCVZg87EWbFxFUZC<br>1Q5qfq1tC7ptd5bGWimbCkJT1hp8v9eNfu |

causing R2D2 to time out. However, both require malware authors to trade agility and easy deployment for more complex malware, which is the opposite of the trend we and others [21, 168] have observed.

#### 4.5.2 Uncooperative Web App Providers

Web app providers are culpable for hosting malicious content [169]. At the same time, they must weigh freedom of expression while ensuring users adhere to the terms and conditions [170, 171, 169]. Toward usability, web app providers seek to ensure they do not unnecessarily block content where evidence of abuse is unavailable [169]. Therefore, like Twitter (Figure 4.6), we may encounter cases where providers are uncooperative to remediate DDR behavior on their platform. Since web app providers maintain sole control, we are limited to their actions against DDR botnets.

#### 4.5.3 Domain Generation Algorithm Domain Origin Identification

Since most DGA-based malware rely on system information retrieved from APIs to seed DGAs, our approach to inject symbolic data from these APIs means that we can track the symbolic data and identify the origin of the DGA web app account. However, R2D2 does not consider the rare cases where the seed is hard-coded into the malware. Malware authors generally avoid this approach because it makes predicting future candidate domains easier [122]. An option to handle this hard coded seeds is re-execution allowing multiple domains to be used and exposed which reveal the integration of a DGA.

#### 4.5.4 A Subtler Case of Implicit Flows

In subsection 4.2.2, we discussed the common case of implicit flows that we consider. However, a subtler case occurs where  $x$  is initialized to a default value before the conditional branch (Algorithm 1, line 3). In this example, we assume there is no `else` condition, meaning  $x$  is either the default value, or the new value (line 4) if the condition has been

met. If the condition at line 3 is not met, R2D2 would not associate  $t_a$  to  $x$  and fail to propagate the tag. If DDR-based malware use this approach, then the default value is used and it likely corresponds to a hard-coded domain name. This is plausible if the malware authors include a backup domain in the even that the web app endpoint is not reachable. However, we have not see this subtler case of implicit flow in practice.

## 4.6 Conclusion

This chapter comprehensively studied the under-explored DDR technique using R2D2, our measurement pipeline, to analyze 100k malware spanning back 5 years, revealing 10,170 DDR-based malware from 154 families. R2D2 also revealed the type of encoding used, providing authorities with rapid means to decode C&C server domains, with String Parsing and Base64 being the most common. We reported all of our findings to web app providers, and they confirmed them and took action against the 9,155 DDRs (90% of DDR-based malware discovered). Of the remainder, web app providers previously took down the accounts for 774 malware, and we are awaiting a response concerning 241 DDRs.

## CHAPTER 5

### CONCLUSION AND FUTURE WORK

The first step to monitor and counteract botnets is to study their C&C infrastructures. However, these infrastructures consist of victim system information (e.g., network logs, etc.), malware binaries or memory image snapshots containing malware payloads, proxy servers, C&C servers, etc. Authorities resort to intensive analyses of the available components, giving plenty of time for C&C orchestrators to cover their tracks. This dissertation shows that malware binary logic can be reused to enable automated and scalable opportunities for *rapid* botnet disruption and takedown. Underpinning the research solutions discussed herein are fundamental techniques for binary program instrumentation, binary functionality analysis, and communication protocol inference. To recap what this dissertation proposes, we provide a discussion through our Goals (Section 5.1), Challenges (Section 5.2), and Solutions (Section 5.3). We then end with a discussion of future work (Section 5.4).

#### 5.1 Goals

The current botnet monitoring state-of-the-art uses cross-domain analysis of numerous C&C infrastructure components and malware variants via static and dynamic analysis techniques. However, these early approaches required context switching and are often prone to human error. Furthermore, being prohibitively tedious, when authorities eventually make significant progress in their investigations, C&C orchestrators are likely to have employed defensive evasions tactics, including C&C server migration. In this dissertation, our goal was to design and implement practicable solutions that submit to single domain analysis towards effectiveness and scalability. A more detailed list of our goals follow:

### 5.1.1 Scalable Malware Analysis

To explore our goals, we performed systematic malware studies. These studies are scalable, reproducible, and provides the requisite information to identify and leverage malware binary logic for C&C monitoring, infiltration, disruption, or takedown. Thus, our first goal was to design and implement malware analysis frameworks that focuses on targeted exploration (to reveal reusable malware binary logic) instead of code coverage. Reducing the scope of exploration to only relevant paths ensured our approach is scalable. We also evaluated and validated our framework using malware captured in the wild over 15 years. This ensured our findings sheds light on the evolution of reusable malware logic in the malware threat environment. Toward reproducibility, we made relevant source code available to the security community.

### 5.1.2 Reusable Malware Logic

To submit to single-domain analysis and practicability, we aimed to identify reusable malware logic for botnet counteraction. Based on our manual reverse engineering and reports from academic and industry experts, we identified three types of reusable malware logic:

1. **Over-Permissioned Protocols:** These are standardized protocols that provide file transfer, data storage, and message-based communications. However, they are also feature-rich and provide unfettered access to the C&C server beyond the subset of features implemented by a given client (bot). For example, if a malware author integrates the File Transfer Protocol (FTP) into their malware but only specifies that the malware *PUT* files from the victim system onto the C&C server, authorities can leverage this to do the same. However, since the entire protocol is *baked* in, authorities can leverage FTP to also *GET* files from the C&C server since that capability is inherent in FTP implementations. Secondly, FTP requires client-side authentication to access the C&C server, meaning the malware stores this information so that bots

can authenticate to their C&C server. Identifying these authentication details in the malware allows authorities to access the C&C server under the guise of normal bot operations.

2. **C&C Monitoring Capabilities:** To infer the C&C server's composition and contents, we analyzed the malware to understand what types of capabilities are exercised on the victim system. Notably, some of these capabilities result in data being sent back to the C&C server. For example, password stealers target victim system software to retrieve user credentials. These credentials are aggregated in victim system memory before being exfiltrated to the C&C server. Identifying exfiltrated data formats and types reveals the type of information stored on the C&C server.
3. **Hiding C&C Rendezvous Points in Plain Sight:** C&C orchestrators generally maintain control of all the components of their C&C infrastructures. Recently, some C&C orchestrators have begun using web applications to hide C&C server rendezvous points (i.e., encoded domain names or IPs). Since these web applications are public, the C&C orchestrators post messages or upload files accessible by all web app users, but encoding masquerades the true intent of posted messages allowing them to hide in plain sight. However, with this deception comes a loss of control. Now, web applications providers control how malware resolves their C&C server domains or IPs. If these hidden rendezvous points are identified, authorities can dismantle botnets in collaboration with web application providers.

### 5.1.3 Validating our Approach

We analyzed 300k Windows malware binaries captured within the last 15 years to validate our frameworks. Our goal is to perform the analysis in a scalable manner and prove that the malware logic of interest is reusable and effectively enables botnet disruptions and takedowns.

## 5.2 Challenges

We encountered several challenges while aiming to achieve our goals. For clarity, we subset our challenges in their corresponding goals.

### 5.2.1 Scalable Malware Analysis

Submitting to a single-domain (malware-only) analysis limits data available that may be relevant to botnet counteraction. So, we must ensure that our malware-only analysis is effective. However, modern malware often employs sophisticated obfuscation of packing techniques that impedes analysis and inhibits large-scale studies. Worse still, malware also uses anti-analysis and defensive evasion strategies that make any analyses prohibitively tedious and sometimes error-prone when malware operations are not fully executed. Based on these challenges, pursuing sole static analysis techniques is not an option, and dynamic analysis approaches will limit malware exploration. We opt for concolic analysis to more directly explore the malware. Now, we are inevitably faced with path explosion, making the analyses intractable. Thus, we develop solutions grounded in fundamental techniques for binary program analysis to ensure tractability.

### 5.2.2 Reusable Malware Logic

Identifying reusable malware logic is difficult because what can be leveraged for one botnet may not be useful for other botnets. Furthermore, as we discovered, malware authors can use different code implementations to accomplish the same task. Thus, finding reusable and scalable malware logic and a generalizable means of identification is not trivial.

### 5.2.3 Validating our Approach

Validating our techniques means affecting the operations of C&C infrastructures. This undoubtedly introduces some ethical concerns. However, we follow the precedence estab-

lished in previous works [116, 17, 117, 88] while exposing the weaknesses that make C&C servers vulnerable to infiltration. Moreover, we maintain that any access we do achieve to C&C infrastructure components avoids any claim of tortious interference, as described in *Mouton vs. VC3* [119]. Furthermore, in more overt means of counteractions, we were required to collaborate with web applications providers, whose platforms were being unknowingly abused by malicious actors. Since any user, malicious or benign, agrees to adhere to acceptable use policy requirements, web application providers can remedy any perceived violation without breaching the boundaries outlined in their policies.

### **5.3 Solutions and Results**

This dissertation presented C3PO [21] and R2D2 [22], measurement pipelines that studied the (1) evolution of over-permissioned protocols in 200k malware spanning 15 years and (2) under-explored DDR technique in 100k malware spanning 5 years.

#### 5.3.1 C3PO

C3PO identified 62,202 over-permissioned bots across 8,512 families identifying infiltration vectors that allow C3PO to spoof bot-to-C&C communication. C3PO also identified 443,905 C&C monitoring capabilities, which reveal the composition and contents of the C&C server to guide monitoring post infiltration. We deployed C3PO on two bots with live C&C servers validating its ability to identify over-permissioned protocols, infiltrate C&C servers, and leverage C&C monitoring capabilities to achieve covert monitoring. C3PO also identified over 2500 files containing victim information, additional malicious payloads, exploitation scripts, and stolen credentials, providing legally admissible evidence to engender counteraction attempts. Armed with C3PO, authorities can pursue disruptions and takedowns of over-permissioned protocol-based botnets.



### 5.3.2 R2D2

R2D2 targets the disruption and takedown of DDR-based botnets. During its analysis of 100k malware, R2D2 revealed 10,170 DDR malware from 154 families. R2D2 also revealed the type of encoding used, providing authorities with rapid means to decode C&C server domains, with String Parsing and Base64 being the most common. I reported all of our findings to web app providers, and they confirmed them and took action against the 9,155 DDRs (90% of DDR malware discovered). Of the remainder, web app providers previously took down the accounts for 774 malware, and we are awaiting a response concerning 241 DDRs. This dissertation demonstrates that malware logic can be reused to enable botnet disruption and takedown.

## **5.4 Future Work**

Our future work can be broadly categorized as (1) deceptively responding to C&C servers toward in-depth investigations enabling botnet counteraction and (2) the further exploration of web application abuse to enable C&C infrastructures.

1. C3PO (Chapter 3) provides the means to covertly infiltrate the C&C server under the guise of normal bot operations. Instead of masquerading as a trusted bot, can we instead respond deceptively to C&C servers on behalf of trusted bots to delay discovery and enable more in-depth investigations of live and active C&C servers? The ability to do so can provide more actionable intelligence like the scale of the botnet, the locale of operations, other affected peers in the botnet, etc., towards counteraction.
2. Inspired by R2D2 (Chapter 4), we look to other ways that malware abuse web applications. Specifically, we aim to discover if malware relies on web applications as C&C servers rather than just a pivoting to identify C&C server locations. We aim to understand the scope of web applications abuse across the entire malware operational landscape.

## REFERENCES

- [1] *New action to combat ransomware ahead of u.s. elections*, <https://blogs.microsoft.com/on-the-issues/2020/10/12/trickbot-ransomware-cyberthreat-us-elections/>, [Accessed: 2020-12-10].
- [2] C. Rossow *et al.*, “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.
- [3] B. Krebs, *U.s. cyber command behind trickbot tricks*, <https://krebsonsecurity.com/2020/10/report-u-s-cyber-command-behind-trickbot-tricks/>, [Accessed: 2020-08-22].
- [4] Y. Nadji, M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee, “Beheading Hydras: Performing Effective Botnet Takedowns,” in *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, Berlin, Germany, Oct. 2013, pp. 121–132.
- [5] R. Wainwright and F. J. Cilluffo, *Responding to cybercrime at scale: Operation avalanche — a case study*, <http://www.jstor.org/stable/resrep20752>.
- [6] *New action to disrupt world’s largest online criminal network*, <https://blogs.microsoft.com/on-the-issues/2020/03/10/necurs-botnet-cyber-crime-disrupt/>, [Accessed: 2020-03-12].
- [7] V. Le Pochat *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.
- [8] *Avast and French police take over malware botnet and disinfect 850,000 computers*, <https://www.zdnet.com/article/avast-and-french-police-take-over-malware-botnet-and-disinfect-850000-computers/>, [Accessed: 2020-03-29].
- [9] W. Sebastian 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.
- [10] *Zeus-p2p monitoring and analysis*, [https://www.cert.pl/wp-content/uploads/2015/12/2013-06-p2p-rap\\_en.pdf](https://www.cert.pl/wp-content/uploads/2015/12/2013-06-p2p-rap_en.pdf), [Accessed: 2020-12-10].

- [11] B. B. Kang *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.
- [12] D. Andriessse, C. Rossow, and H. Bos, “Reliable Recon in Adversarial Peer-to-peer Botnets,” in *Proceedings of the Internet Measurement Conference (IMC)*, Tokyo, Japan, Oct. 2015, pp. 129–140.
- [13] G. Gu, V. Yegneswaran, P. Porras, J. Stoll, and W. Lee, “Active Botnet Probing to Identify Obscure Command and Control Channels,” in *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2009, pp. 241–253.
- [14] C. Zuo, Q. Zhao, and Z. Lin, “Authscope: Towards Automatic Discovery of Vulnerable Authorizations in Online Services,” in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*, Dallas, TX, Oct. 2017, pp. 799–813.
- [15] 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.
- [16] 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.
- [17] Z. Durumeric, E. Wustrow, and J. A. Halderman, “ZMap: Fast Internet-wide Scanning and its Security Applications,” in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 605–620.
- [18] B. Stone-Gross *et al.*, “Your Botnet is my Botnet: Analysis of a Botnet Takeover,” in *Proceedings of the 16th ACM conference on Computer and Communications Security*, 2009, pp. 635–647.
- [19] *Kelihos/hlux botnet returns with new techniques*, <https://securelist.com/kelihoshlux-botnet-returns-with-new-techniques/32021/>, [Accessed: 2020-12-10].
- [20] I. Arghire, *Trickbot botnet survives takedown attempt*, <https://www.securityweek.com/trickbot-botnet-survives-takedown-attempt>, [Accessed: 2020-12-10].
- [21] J. Fuller *et al.*, “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.

- [22] J. Fuller *et al.*, “Is that Malware Reading Twitter? Towards Understanding and Preventing Dead Drop Resolvers on Public Web Apps,” in *In Submission in 2023 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2023.
- [23] L. Böck, E. Vasilomanolakis, M. Mühlhäuser, and S. Karuppayah, “Next Generation P2P Botnets: Monitoring Under Adverse Conditions,” in *Proceedings of the 21th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Crete, Greece, Sep. 2018, pp. 511–531.
- [24] S. Karuppayah, L. Böck, T. Grube, S. Manickam, M. Mühlhäuser, and M. Fischer, “Sensorbuster: On Identifying Sensor Nodes in P2P Botnets,” in *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES)*, Reggio Calabria, Italy, Oct. 2017, pp. 1–6.
- [25] S. Karuppayah, M. Fischer, C. Rossow, and M. Mühlhäuser, “On Advanced Monitoring in Resilient and Unstructured P2P Botnets,” IEEE, 2014, pp. 871–877.
- [26] J. Caballero, P. Poosankam, C. Kreibich, and D. Song, “Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering,” in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009, pp. 621–634.
- [27] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M., “Unexpected Means of Protocol Inference,” in *Proceedings of the Internet Measurement Conference (IMC)*, Rio de Janeiro, Brazil, Oct. 2006, pp. 313–326.
- [28] W. Cui, J. Kannan, and H. J. Wang, “Discoverer: Automatic Protocol Reverse Engineering from Network Traces,” in *Proceedings of the 26th USENIX Security Symposium (Security)*, Vancouver, BC, Canada, Aug. 2017, pp. 1–14.
- [29] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer, “Dynamic Application-layer Protocol Analysis for Network Intrusion Detection,” in *Proceedings of the 15th USENIX Security Symposium (Security)*, Vancouver, Canada, Jul. 2006, pp. 257–272.
- [30] J. Caballero, H. Yin, Z. Liang, and D. Song, “Polyglot: Automatic Extraction of Protocol Message Format Using Dynamic Binary Analysis,” in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, Nov. 2007, pp. 317–329.
- [31] Z. Lin, X. Jiang, D. Xu, and X. Zhang, “Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution,” in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, vol. 8, San Diego, CA, Feb. 2008, pp. 1–15.

- [32] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol Specification Extraction," in *Proceedings of the 30th IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 2009, pp. 110–125.
- [33] D. Binkley, N. Gold, and M. Harman, "An Empirical Study of Static Program Slice Size," 2, vol. 16, ACM, 2007, 8–es.
- [34] D. Lucia, D. Lucca, *et al.*, "Software salvaging based on conditions," in *Proceedings 1994 International Conference on Software Maintenance*, IEEE, 1994, pp. 424–433.
- [35] K. B. Gallagher, "Using program slicing in software maintenance," Ph.D. dissertation, University of Maryland, Baltimore County, 1990.
- [36] M. Weiser and J. Lyle, "Experiments on slicing-based debugging aids," in *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, 1986, pp. 187–197.
- [37] D. Binkley, "The application of program slicing to regression testing," *Information and software technology*, vol. 40, no. 11-12, pp. 583–594, 1998.
- [38] A. De Lucia, A. R. Fasolino, and M. Munro, "Understanding function behaviors through program slicing," in *WPC'96. 4th Workshop on Program Comprehension*, IEEE, 1996, pp. 9–18.
- [39] M. Harman, R. Hierons, C. Fox, S. Danicic, and J. Howroyd, "Pre/post conditioned slicing," in *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001*, IEEE, 2001, pp. 138–147.
- [40] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural Slicing Using Dependence Graphs," in *Proceedings of the 1988 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, vol. 12, Atlanta, GA, Jun. 1988, pp. 26–60.
- [41] M. Weiser, "Program slicing," *IEEE Transactions on software engineering*, no. 4, pp. 352–357, 1984.
- [42] J. Jaffar, V. Murali, J. A. Navas, and A. E. Santosa, "Path-sensitive Backward Slicing," in *Proceedings of the International Static Analysis Symposium (ISAS)*, Springer, 2012, pp. 231–247.
- [43] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: A Tool for Change Impact Analysis of Java Programs," ACM, 2004, pp. 432–448.

- [44] V. Srinivasan and T. Reps, “An Improved Algorithm for Slicing Machine Code,” 10, vol. 51, ACM, 2016, pp. 378–393.
- [45] 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.
- [46] 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.
- [47] 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.
- [48] J. C. King, “Symbolic Execution and Program Testing,” *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.
- [49] 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.
- [50] L. A. Clarke, “A System to Generate Test Data and Symbolically Execute Programs,” 3, IEEE, 1976, pp. 215–222.
- [51] 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.
- [52] K. Kim *et al.*, “J-force: Forced Execution on Javascript,” in *Proceedings of the 26th international conference on World Wide Web*, 2017, pp. 897–906.
- [53] 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. 2011, pp. 1849–1866.
- [54] R. Brotzman, S. Liu, D. Zhang, G. Tan, and M. Kandemir, “Casym: Cache aware symbolic execution for side channel detection and mitigation,” in *CaSym: Cache Aware Symbolic Execution for Side Channel Detection and Mitigation*, IEEE, 2019.
- [55] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, “Firmallice-automatic detection of authentication bypass vulnerabilities in binary firmware.”

in *Proceedings of the 2015 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2015.

- [56] S. Y. Chau *et al.*, “Symcerts: Practical Symbolic Execution for Exposing Noncompliance in X. 509 Certificate Validation Implementations,” in *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2017, pp. 503–520.
- [57] L. Martignoni, S. McCamant, P. Poosankam, D. Song, and P. Maniatis, “Path-exploration lifting: Hi-fi tests for lo-fi emulators,” in *Proceedings of the 17th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, London, UK, Mar. 2012.
- [58] Y. Shoshitaishvili *et al.*, “Mechanical Phish: Resilient Autonomous Hacking,” in *Proceedings of the 39th IEEE Symposium on Security and Privacy (S&P)*, vol. 16, San Francisco, CA, May 2018, pp. 12–22.
- [59] C. Zuo and Z. Lin, “Smartgen: Exposing Server Urls of Mobile Apps with Selective Symbolic Execution,” in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 867–876.
- [60] O. Alrawi *et al.*, “Forecasting Malware Capabilities From Cyber Attack Memory Images,” in *Proceedings of the 30th USENIX Security Symposium (Security)*, Virtual Conference, Aug. 2021.
- [61] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer, “Behavior-based Spyware Detection,” in *USENIX Security Symposium (Security)*, 2006, p. 694.
- [62] 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.
- [63] 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.
- [64] 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.
- [65] Y. Aafer, W. Du, and H. Yin, “Droidapiminer: Mining Api-level Features for Robust Malware Detection in Android,” in *International Conference on Security and Privacy in Communication Systems*, Springer, 2013, pp. 86–103.

- [66] 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.
- [67] 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.
- [68] 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.
- [69] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda, “Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries,” in *2010 IEEE Symposium on Security and Privacy*, IEEE, 2010, pp. 29–44.
- [70] G. Anthes, “Security in the Cloud,” *Communications of the ACM*, vol. 53, no. 11, pp. 16–18, 2010.
- [71] Y. Chen, V. Paxson, and R. H. Katz, “What’s New About Cloud Computing Security,” *University of California, Berkeley Report No. UCB/EECS-2010-5 January*, vol. 20, no. 2010, pp. 2010–5, 2010.
- [72] F. Maggi and S. Zanero, “Rethinking security in a cloudy world,” *Politecnico di Milano, Tech. Rep. TR-2010-11*, 2010.
- [73] K. Clark, M. Warnier, and F. M. Brazer, “BOTCLOUDS: The Future of Cloud-based Botnets?” In *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER)*, Noordwijkerhout, Netherlands, May 2011.
- [74] 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)*, IEEE, 2014.
- [75] 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.
- [76] 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.



- [77] 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.
- [78] M. Torkashvan and H. Haghghi, “CB2C: A Cloud-Based Botnet Command and Control,” *Indian Journal of Science and Technology*, vol. 8, no. 22, p. 1, 2015.
- [79] W. Lu, M. Miller, and L. Xue, “Detecting Command and Control Channel of Botnets in Cloud,” in *International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*, Springer, 2017, pp. 55–62.
- [80] S. Zhao, P. P. Lee, J. C. 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 Annual Computer Security Applications Conference (ACSAC)*, 2012, pp. 119–128.
- [81] *Netskope Threat Research Reveals More Than Two-Thirds of Malware Downloads Came From Cloud Apps in 2021*, <https://www.netskope.com/press-releases/netskope-threat-research-reveals-more-than-two-thirds-of-malware-downloads-came-from-cloud-apps-in-2021>, [Accessed: 2022-03-12].
- [82] *Attack matrix for enterprise*, <https://attack.mitre.org/>, [Accessed: 2021-11-06].
- [83] *Trickbot botnet survives takedown attempt, but microsoft sets new legal precedent*, <https://www.zdnet.com/article/trickbot-botnet-survives-takedown-attempt-but-microsoft-sets-new-legal-precedent/>, [Accessed: 2020-12-10].
- [84] *An update on disruption of trickbot*, <https://blogs.microsoft.com/on-the-issues/2020/10/20/trickbot-ransomware-disruption-update>, [Accessed: 2020-12-10].
- [85] S. Banescu and A. Pretschner, “A Tutorial on Software Obfuscation,” vol. 108, Elsevier, 2018, pp. 283–353.
- [86] *Carbanak APT: The Great Bank Robbery*, [https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08064518/Carbanak\\_APT\\_eng.pdf](https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08064518/Carbanak_APT_eng.pdf), [Accessed: 2020-04-16].
- [87] A. Mandal, *Thick Client Application Security*, [http://www.infosecwriters.com/Papers/AMandal\\_Thick\\_Client\\_Application\\_Security.pdf](http://www.infosecwriters.com/Papers/AMandal_Thick_Client_Application_Security.pdf), [Accessed: 2020-04-18].
- [88] O. Alrawi, C. Zuo, R. Duan, R. P. Kasturi, Z. Lin, and B. Saltaformaggio, “The Betrayal at Cloud City: An Empirical Analysis of Cloud-based Mobile Backends,” in *Proceedings of the 28th USENIX Security Symposium (Security)*, Santa Clara, CA, Aug. 2019, pp. 551–566.

- [89] *Command and Control Used in Sanny APT Attacks Shut Down*, <https://threatpost.com/command-and-control-used-sanny-apt-attacks-shut-down-032213/77658/>, [Accessed: 2021-01-09].
- [90] *Sanny Malware Updates Delivery Method*, <https://threatpost.com/sanny-malware-updates-delivery-method/130803/>, [Accessed: 2021-01-09].
- [91] B. Cheng *et al.*, “Towards Paving the way for Large-Scale Windows Malware Analysis: Generic Binary Unpacking with Orders-of-Magnitude Performance Boost,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 395–411.
- [92] X. Ugarte-Pedrero, D. Balzarotti, I. Santos, and P. G. Bringas, “SoK: Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-time Packers,” in *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, May 2015, pp. 659–673.
- [93] Mandiant, *APT1: Exposing One of China’s Cyber Espionage Units*, <https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf>, [Accessed: 2020-05-23].
- [94] M. D. Brown and S. Pande, “CARVE: Practical Security-focused Software Debloating using Simple Feature Set Mappings,” in *Proceedings of the 3rd ACM Workshop on Forming an Ecosystem Around Software Transformation (FEAST)*, London, United Kingdom, 2019, pp. 1–7.
- [95] *Microsoft Documentation*, <https://docs.microsoft.com/en-us/>, [Accessed: 2021-01-09].
- [96] C. Kalt, “Internet relay chat: Client protocol,” RFC Editor, RFC 2812, Apr. 2000.
- [97] *MRFC 1350 - The TFTP Protocol*, <https://tools.ietf.org/html/rfc1350>, [Accessed: 2021-01-09].
- [98] *MySQL Documentation*, <https://dev.mysql.com/doc/>, [Accessed: 2021-01-09].
- [99] *MongoDB C Driver*, <https://docs.mongodb.com/drivers/c/>, [Accessed: 2021-01-09].
- [100] *Malpedia: Free and Open Malware Reverse Engineering Resource offered by Fraunhofer FKIE*, <https://malpedia.caad.fkie.fraunhofer.de>, [Accessed: 2021-11-06].
- [101] G. Hunt and D. Brubacher, “Detours: Binary Interception of Win32 Functions,” in *Proceedings of the 3rd USENIX Windows NT Symposium*, Seattle, WA, Jul. 1999.

- [102] Y. Shoshitaishvili *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, pp. 138–157.
- [103] 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.
- [104] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, “Avclass: A Tool for Massive Malware Labeling,” in *Proceedings of the 19th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Evry, France, Sep. 2016, pp. 230–253.
- [105] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis, “A Lustrum of Malware Network Communication: Evolution and Insights,” in *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2017, pp. 788–804.
- [106] P. Kotzias, L. Bilge, and J. Caballero, “Measuring PUP Prevalence and PUP Distribution through Pay-Per-Install Services,” in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 739–756.
- [107] X. Mi *et al.*, “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.
- [108] 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.
- [109] R. Perdisci, W. Lee, and N. Feamster, “Behavioral Clustering of HTTP-based Malware and Signature Generation using Malicious Network Traces,” in *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, Apr. 2010.
- [110] R. F. M. Dollah, M. Faizal, F. Arif, M. Z. Mas’ud, and L. K. Xin, “Machine Learning for HTTP Botnet Detection Using Classifier Algorithms,” vol. 10, *Universiti Teknikal Malaysia Melaka*, 2018, pp. 27–30.
- [111] T. Nelms, R. Perdisci, and M. Ahamad, “Execscent: Mining for new C&C Domains in Live Networks with Adaptive Control Protocol Templates,” in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 589–604.

- [112] A. Oprea, Z. Li, R. Norris, and K. Bowers, “Made: Security Analytics for Enterprise Threat Detection,” in *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*, 2018.
- [113] *New Chrome Password Stealer Sends Stolen Data to a MongoDB Database*, <https://www.bleepingcomputer.com/news/security/new-chrome-password-stealer-sends-stolen-data-to-a-mongodb-database/>, [Accessed: 2020-02-06].
- [114] X. Lin, *Expiro malware is back and even harder to remove*, <https://www.mcafee.com/blogs/other-blogs/mcafee-labs/expiro-infects-encrypts-files-to-complicate-repair/>, [Accessed: 2020-08-14].
- [115] *Peid*, <https://www.aldeid.com/wiki/PEiD>, [Accessed: 2021-01-11].
- [116] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, you, get off of my Cloud: Exploring Information Leakage in Third-Party Compute Clouds,” in *Proceedings of the 16th ACM conference on Computer and Communications Security*, 2009, pp. 199–212.
- [117] F. Li *et al.*, “You’ve got Vulnerability: Exploring Effective Vulnerability Notifications,” in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1033–1050.
- [118] A. J. Burstein, “Conducting Cybersecurity Research Legally and Ethically,” in *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, vol. 8, San Francisco, CA, Apr. 2008, pp. 1–8.
- [119] *Moulton vs. VC3*, <http://www.internetlibrary.com/pdf/Moulton-VC3.pdf>, [Accessed: 2020-08-14].
- [120] *Latest steam malware shows signs of rat activity*, <https://blog.malwarebytes.com/cybercrime/2016/03/latest-steam-malware-shows-sign-of-rat-activity/>, [Accessed: 2020-08-20].
- [121] *VirusTotal*, <https://www.virustotal.com/>, [Accessed: 2022-1-5].
- [122] M. Antonakakis *et al.*, “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.
- [123] *Donot team leverages new modular malware framework in south asia*, <https://www.netscout.com/blog/asert/donot-team-leverages-new-modular-malware-framework-south-asia>, [Accessed: 2020-08-22].

- [124] *Fast Flux Networks Working and Detection*, <https://resources.infosecinstitute.com/topic/fast-flux-networks-working-detection-part-1>, [Accessed: 2022-03-12].
- [125] *Dynamic Resolution: Fast Flux DNS*, <https://attack.mitre.org/techniques/T1568/001/>, [Accessed: 2022-03-12].
- [126] *Fast Flux networks: What are they and how do they work?* <https://www.welivesecurity.com/2017/01/12/fast-flux-networks-work>, [Accessed: 2022-03-12].
- [127] *Dynamic Resolution: DNS Calculation*, <https://attack.mitre.org/techniques/T1568/003/>, [Accessed: 2022-03-12].
- [128] *Whois Numbered Panda*, <https://www.crowdstrike.com/blog/whois-numbered-panda/>, [Accessed: 2022-03-12].
- [129] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, “A Comprehensive Measurement Study of Domain Generating Malware,” in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 263–278.
- [130] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, “Exposure: Finding Malicious Domains Using Passive DNS Analysis,” in *NDSS*, 2011, pp. 1–17.
- [131] *Hackers buying space from major cloud providers to distribute malware*, <https://www.securitymagazine.com/articles/96900-hackers-buying-space-from-major-cloud-providers-to-distribute-malware>, [Accessed: 2022-03-12].
- [132] *Cloud and Threat Report: Cloudy with a Chance of Malware*, <https://www.netskope.com/blog/cloud-and-threat-report-cloudy-with-a-chance-of-malware>, [Accessed: 2022-03-12].
- [133] R. P. Kasturi *et al.*, “TARDIS: Rolling back the Clock on CMS-Targeting Cyber Attacks,” in *2020 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2020, pp. 1156–1171.
- [134] R. P. Kasturi *et al.*, “Mistrust Plugins You Must: A Large-Scale Study Of Malicious Plugins In WordPress Marketplaces,” in *Proceedings of the 31th USENIX Security Symposium (Security)*, Boston, MA, Aug. 2022.
- [135] *Read The Manual: A Guide to the RTM Banking Trojan*, <https://www.welivesecurity.com/wp-content/uploads/2017/02/Read-The-Manual.pdf>, [Accessed: 2022-03-12].
- [136] *APT17: Hiding in Plain Sight - FireEye and Microsoft Expose Obfuscation Tactic*, <https://www.fireeye.com/current-threats/apt-groups/rpt-apt17.html>, [Accessed: 2021-02-21].

- [137] *Web service*, <https://attack.mitre.org/techniques/T1102/>, [Accessed: 2021-11-06].
- [138] *OPERATION GHOST. The Dukes aren't back — they never left*, [https://www.welivesecurity.com/wp-content/uploads/2019/10/ESET\\_Operation\\_Ghost\\_Dukes.pdf](https://www.welivesecurity.com/wp-content/uploads/2019/10/ESET_Operation_Ghost_Dukes.pdf), [Accessed: 2022-02-26].
- [139] *Casbaneiro: Dangerous Cooking with a Secret Ingredient*, <https://www.welivesecurity.com/2019/10/03/casbaneiro-trojan-dangerous-cooking/>, [Accessed: 2022-03-06].
- [140] *The Dropping Elephant - Aggressive Cyber-Espionage in the Asian Region*, <https://securelist.com/the-dropping-elephant-actor/75328/>, [Accessed: 2022-03-06].
- [141] *Analyzing malware by API calls*, <https://blog.malwarebytes.com/threat-analysis/2017/10/analyzing-malware-by-api-calls/>, [Accessed: 2022-03-06].
- [142] F. Bellard, “QEMU, a Fast and Portable Dynamic Translator,” in *USENIX Annual Technical Conference, FREENIX Track*, California, USA, vol. 41, 2005, pp. 10–5555.
- [143] *Getting Started with Winsock*, <https://docs.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock>, [Accessed: 2021-11-06].
- [144] *About WinINet*, <https://docs.microsoft.com/en-us/windows/win32/wininet/about-wininet>, [Accessed: 2022-03-12].
- [145] *About WinHTTP*, <https://docs.microsoft.com/en-us/windows/win32/winhttp/about-winhttp>, [Accessed: 2022-03-12].
- [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] J. Gao and S. S. Lumetta, “Loop Path Reduction by State Pruning,” in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2018, pp. 838–843.

- [150] 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.
- [151] H. Asghari, M. Ciere, and M. J. Van Eeten, “Post-mortem of a Zombie: Conficker Cleanup after Six Years,” in *Proceedings of the 24th USENIX Security Symposium (Security)*, Washington, DC, Aug. 2015, pp. 1–16.
- [152] *Double Dragon - APT41, a Dual Espionage and Cyber Crime Operation*, <https://content.fireeye.com/apt-41/rpt-apt41>, [Accessed: 2022-03-06].
- [153] *Monsoon – Analysis of an APT Campaign*, <https://www.forcepoint.com/sites/default/files/resources/files/forcepoint-security-labs-monsoon-analysis-report.pdf>, [Accessed: 2022-03-06].
- [154] *Understanding the Patchwork Cyberespionage Group*, <https://documents.trendmicro.com/assets/tech-brief-untangling-the-patchwork-cyberespionage-group.pdf>, [Accessed: 2022-03-06].
- [155] *Bronze Butler Targets Japanese Enterprises*, <https://www.secureworks.com/research/bronze-butler-targets-japanese-businesses>, [Accessed: 2022-03-06].
- [156] *Pony’s C&C Servers Hidden Inside the Bitcoin Blockchain*, <https://research.checkpoint.com/2019/ponys-cc-servers-hidden-inside-the-bitcoin-blockchain/>, [Accessed: 2022-03-06].
- [157] *Multigrain - Point of Sale Attackers Make an Unhealthy Addition to the Pantry*, [https://www.fireeye.com/blog/threat-research/2016/04/multigrain\\_pointo.html](https://www.fireeye.com/blog/threat-research/2016/04/multigrain_pointo.html), [Accessed: 2021-11-06].
- [158] *Gustuff Banking Botnet Targets Australia*, <https://blog.talosintelligence.com/2019/04/gustuff-targets-australia.html>, [Accessed: 2022-03-06].
- [159] *Biopass RAT: New Malware Sniffs Victims via Live Streaming*, [https://www.trendmicro.com/en\\_us/research/21/g/biopass-rat-new-malware-sniffs-victims-via-live-streaming.html](https://www.trendmicro.com/en_us/research/21/g/biopass-rat-new-malware-sniffs-victims-via-live-streaming.html), [Accessed: 2021-11-06].
- [160] *Malware Campaign Targets South Korean Banks*, <https://blog.trendmicro.com/trendlabs-security-intelligence/malware-campaign-targets-south-korean-banks-uses-pinterest-as-cc-channel/>, [Accessed: 2021-11-06].
- [161] *A System for Detecting Software Similarity*, <https://theory.stanford.edu/~aiken/moss/>, [Accessed: 2022-02-26].

- [162] *The Malicious Use of Pastebin*, <https://www.fortinet.com/blog/threat-research/malicious-use-of-pastebin>, [Accessed: 2022-03-12].
- [163] T. Taniguchi, H. Griffioen, and C. Doerr, “Analysis and Takeover of the Bitcoin-Coordinated Pony Malware,” in *Proceedings of the 16th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Hong Kong, China, Jun. 2021.
- [164] *Bitcoin abuse database*, <https://www.bitcoinabuse.com/>, [Accessed: 2022-2-06].
- [165] J. Li, Z. Lin, J. Caballero, Y. Zhang, and D. Gu, “K-Hunt: Pinpointing Insecure Cryptographic Keys from Execution Traces,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 412–425.
- [166] D. Xu, J. Ming, and D. Wu, “Cryptographic Function Detection in Obfuscated Binaries via Bit-Precise Symbolic Loop Mapping,” in *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2017, pp. 921–937.
- [167] C. Meijer, V. Moonsamy, and J. Wetzels, “Where’s crypto?: Automated identification and classification of proprietary cryptographic primitives in binary code,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 555–572.
- [168] *How malware writers’ laziness is helping one startup predict attacks before they even happen*, <https://www.zdnet.com/article/how-malware-writers-laziness-is-helping-one-startup-predict-attacks-before-they-even-happen/>, [Accessed: 2022-1-16].
- [169] K. Thomas *et al.*, “Sok: Hate, Harassment, and The Changing Landscape of Online Abuse,” in *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2021, pp. 247–267.
- [170] J. A. Pater, M. K. Kim, E. D. Mynatt, and C. Fiesler, “Characterizations of Online Harassment: Comparing Policies Across Social Media Platforms,” in *Proceedings of the 19th international conference on supporting group work*, 2016, pp. 369–374.
- [171] S. Zannettou, J. Blackburn, E. De Cristofaro, M. Sirivianos, and G. Stringhini, “Understanding Web Archiving Services and their (Mis) Use on Social Media,” in *Twelfth International AAAI Conference on Web and Social Media*, 2018.