Is That Malware Reading Twitter? Towards Understanding and Preventing Dead Drop Resolvers

Anonymous Author(s)

ABSTRACT

Malware authors are integrating an entirely novel trick: using internet dead drops to retrieve rendezvous points for their C&C server. Known as Dead Drop Resolvers (DDRs), these malware can migrate C&C servers by simply posting *rendezvous points* (i.e., encoded URLs or IPs) on public internet services. Hiding in plain sight, malware authors manipulate their posted content, so authorities remain unaware of their true intent. Authorities must undergo extensive analysis to de-manipulate the C&C address before taking action. This research aims to study this DDR adoption trend and counteract these threats. We developed R2D2, an automated DDR malware analysis pipeline, and analyzed 100K malware identifying 10,170 DDR malware from 154 families. R2D2 also revealed the DDR de-manipulation schemes, providing authorities with a rapid means to decode C&C addresses. We reported our findings to service providers, who confirmed and took action against 9,155 DDRs (90% of DDR malware discovered).

CCS CONCEPTS

- **Security and privacy → Malware and its mitigation**;

KEYWORDS

Malware; Internet Dead Drops; Botnet Counteraction

1 INTRODUCTION

Internet *dead drops* allow malware, known as Dead Drop Resolvers (DDRs), to dynamically *resolve* their C&C server IP or URL. These dead drops (D2) include everything from social media networks (e.g., Twitter) to data hosting platforms (e.g., Dropbox) to bitcoin transactions on the blockchain. These services allow anonymous access to user-created content, and network traffic to these services appears benign to firewall filters [1], [2]. Importantly, botnet operators *manipulate* (i.e., encrypt or encode) their posts to prevent service providers from attributing the posts to botnet activity. Instead, service providers must wait for authorities to report content on their platforms related to botnet infections. A recent discovery demonstrated this and took down one botnet abusing the blockchain [3]–[5], which highlighted the extensive manual analysis required to identify DDRs, *de-manipulate* the C&C server address (called "rendezvous point"), and attempt content revocation.

To appreciate the novel challenges of DDR, it is important to consider how it improves upon the prior botnet state-of-the-art. Botnet takedowns rely on recovering C&C addresses for seizure and sinkholing [6]–[11]. Naive malware encode static C&C addresses *in their binary*, but ample research exists to recover manipulated URLs from a binary [12]–[14]. Modern botnets confound takedown attempts by dynamically resolving C&C addresses. Earlier botnets used fast-flux networks [15]–[17] or domain name service (DNS) calculation [18], [19], but these provide only a small range of C&C address options, which makes recovering C&C addresses straightforward [16], [19]. To complicate C&C address recovery, domain generation algorithms (DGA) [20]–[22] can generate an unlimited number of C&C addresses, making it the preferred approach for advanced botnets. Fortunately, because the DGA is available *in the malware binary*, existing techniques can re-implement the DGA to predict all future candidate domains [23]–[26].

DDRs improve upon the prior C&C resolution approaches in several ways. First, C&C addresses are *unpredictable* because, unlike DGAs, no algorithm exists to generate them. Botnet operators simply post a new rendezvous point. Second, DDRs prevent botnet sinkholing. Instead, authorities must rely on service providers to revoke the posted content and disable future posts. Moreover, service providers alone can hardly attribute content on their platforms to C&C rendezvous points (e.g., Figure 1 in §2). Third, C&C addresses are no longer discoverable via malware binary analysis. At best, existing techniques [12], [27]–[29] can tell authorities that the malware is connecting to a benign service on the Internet. Authorities must attribute the content (e.g., accounts, posts, blockchain transactions) to DDR and report to the provider. Unfortunately, our research found some providers *are reluctant* to remove C&C rendezvous points even with proof of its use for DDR (§5.5).

Ideally, service providers can remove the account (analogous to domain seizure), preventing the botnet from leveraging dead drops. Alternatively, if authorities know the correct data manipulation recipe, service providers can *replace* the content with the sinkhole server IP/URL [24], [25], [30]. Interestingly, the malware binary itself *implements a DDR client* — connecting to the D2 service, searching for its content, and fetching the C&C rendezvous point. This prompted our first key insight: Authorities must extract not only the D2 but also the malware's DDR logic to search for and fetch DDR content. After the rendezvous point is fetched, *de-manipulating* requires time-consuming manual analysis [3]–[5]. Surprisingly, our research found individual malware layering up to 4 data manipulation schemes, 22,937 schemes, and an average of 2 schemes per malware (§5.2). This prompted our second key insight: If authorities extract this de-manipulation "recipe", it can be used to verify DDR content on the D2 service or enable rendezvous point replacement and botnet sinkholing.

To explore our key insights, we comprehensively studied DDR adoption in malware in collaboration with <redacted>, a network edge security provider serving over 25% of the Fortune 100. To **R**eveal **R**endezvous Points from **D**ead **D**rops, we develop R2D2, a malware binary analysis framework that automates the effort needed to seize or sinkhole DDR botnets. R2D2 concollicaly explores the malware to *localize the DDR logic*, confirming DDR integration. Then, R2D2 looks for the *hybrid DDR+DGA capability*, which sheds light on a unique malware practice: DDR malware that use DGAs to generate accounts for D2 services. Next, R2D2 moves to identify data manipulation routines in the malware by searching through the symbolic expression derived during concolic analysis. However, we identified a unique challenge whereby R2D2 must isolate only the portion of the expression related to data manipulation to ensure that the matching is effective and accurate. To enable this, R2D2 uses a novel *Input/Output (IO) boundary isolation* technique to isolate routines in the symbolic expression. Using these boundaries, R2D2 *matches symbolic expression* to form a de-manipulation recipe.

We deployed R2D2 on 100K malware captured between 2017 to 2022 and uncovered 10,170 DDR malware from 154 families. R2D2 found that 95% of DDR malware use 1 or more de-manipulation algorithms, an average of 2 per malware, and over 53% use String Parsing with Base64 as their recipe. We also found blockchain services were the most popular category of DDR-enabling services accounting for over 30% of DDR malware in our dataset. Since we lack authority for seizure or sinkholing, we reported all of our findings to the appropriate service providers, who 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 service providers hosting the other 241 DDRs.

2 CHALLENGES AND OPPORTUNITIES

Recently, industry experts have sounded the alarm on malware abusing internet services in novel ways [2], [31]–[33]. Yet, mitigating these threats has been ineffective. To date, MITRE [34] suggests intrusion detection systems for (1) blocking malicious traffic to benign websites or (2) restricting all access (benign or not) to web-based content used by malware [35]. However, authorities can hardly isolate malware-specific traffic to D2s from all other benign traffic. Moreover, the services enabling D2s are rightly used by organizations globally [31], so blanket restrictions are problematic. Using the *razy* incident [36] as an example, we show how authorities could respond rapidly with R2D2.

Challenge 1: "Why is that malware reading Twitter?": *Razy* floods VirusTotal with submissions and retrieves a message from Twitter. The initial *razy* analysts used API calls and network trace analysis to identify contacted domains [36]. Yet, they could not confirm how the malware used the Twitter message, so they focused on its VirusTotal-related actions, which only served as a distraction.

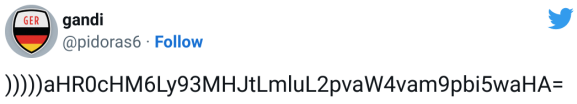

Figure 1: Twitter Message Retrieved by Razy.

R2D2 confirmed *razy* uses Twitter to dynamically resolve its C&C server address. R2D2 uses concolic analysis, which is beneficial for performing a large-scale study for four reasons: (1) R2D2 can localize the DDR logic via data flow analysis (§3.1). (2) Concolic analysis is also necessary *prior to* (1) because R2D2 considers hybrid DDR+DGA techniques (§3.1) in malware [37]. Specifically, our research uncovered 12 DDR malware using DGAs to dynamically generate DDR identities. (3) Symbolic data enables analysis even after botnet operators have deleted the DDR account. (4) The symbolic expressions enable R2D2 to *reverse data manipulation* of the rendezvous point (§3.2) to form the de-manipulation recipe (§3.3).

Challenge 2: "Does that Tweet look suspicious?": The previous *razy* investigation could not prove how the malware used the Twitter post. So, they viewed the Twitter account page and noted the suspicious-looking message (Figure 1). An experienced analyst may infer some form of Base64 encoding. However, investigators have no way to identify data manipulation types automatically, and when faced with multiple manipulation techniques, C&C address recovery is a prohibitively complex task.

Our second key insight (§1) led us to consider the de-manipulation recipe. Since DDR malware often use more than 1 de-manipulation routine, identifying routines in order (i.e., the recipe) enables the recovery of the C&C server address from the rendezvous point. R2D2 identified that *razy* selectively removes preceding characters before applying a Base64 decoder. However, this requires R2D2 to match a known decoding algorithm to the malware's arbitrary implementation of those algorithms — a challenging goal as there are infinitely many implementations of the same decoding logic. To solve this challenge, R2D2 uses symbolic data to represent the Twitter message. As the malware executes, this symbolic data forms an expression that assumes additional values corresponding to the computations performed on the data. Thus, the expression pertains to both the string parsing and Base64 decoding. When the DDR logic is localized, and DDR integration is confirmed in the malware (§3.1), R2D2 takes the symbolic expression generated from the initial analysis and compares it with a set of reference decoding algorithms that are pre-built in R2D2 (§3.2 and §3.3).

Challenge 3: "Sinkhole and seizure from a Tweet?": Botnet orchestrators relinquish some control by using D2s as a trampoline to their C&C server. To seize or sinkhole the *razy* C&C server, authorities must uncover the C&C address by de-manipulating the tweet or requesting that it be removed. They can also *encode* their sinkhole URL in the expected format and replace the rendezvous point.

Therefore, our final goal is to provide authorities with a de-manipulation recipe that they can use to sinkhole or seize the DDR. While R2D2 can compare 2 symbolic expressions for algorithm verification, much of the malware's symbolic expression is unrelated to data de-manipulation. So, R2D2 needs to isolate only the input and output domains that mark the start and end of each de-manipulation algorithm in the malware's symbolic expression before comparing it with our set of reference decoder algorithms. Thus, we develop a novel approach to isolate the domains (§3.2). This results in a de-manipulation *recipe* arming investigators with the means to automatically uncover C&C addresses from rendezvous points. For *razy*, R2D2 recovered `https://w0rm.in/join/join.php`, a hacking forum used to send C&C commands to *razy* bots.

Authorities can collaborate with service providers to take down the reported content — as we did in this research (§5.5). They can also use R2D2's de-manipulation recipe to monitor and extract newly posted rendezvous points. Furthermore, providers can *replace* the rendezvous points with an authority-owned sinkhole URL *manipulated in the correct format.*

3 R2D2'S ANALYSIS PIPELINE

R2D2's workflow proceeds through the following phases: *DDR logic localization* (§3.1) is used to confirm DDR integration in malware. The output of this phase is a symbolic expression containing computations from malware exploration. R2D2 then *isolates de-manipulation boundaries* (§3.2) to segment the symbolic expression for comparison with a set of reference de-manipulation algorithms for *recipe verification* (§3.3).

3.1 DDR Logic Localization

To localize DDR logic, R2D2 uses concolic analysis because concrete exploration allows the malware to unpack itself natively [38]–[41]. Furthermore, since we studied malware captured from 2017, we expect some dead D2 endpoints. So, R2D2 uses modeled network APIs (e.g., `connect`), enabling malware exploration regardless of endpoint liveness. When a network API is invoked, R2D2 injects symbolic return values to trick the malware (e.g., `GetLocaleInfo` retrieves the bot's location before continuing the attack). These modeled APIs allow R2D2 to selectively inject symbolic data to ensure continued malware exploration. Appendix A provides an extendable list of all modeled defensive evasion APIs.

DDR Client Connection. After executing through packing layers and surmounting anti-analysis techniques, R2D2 intercepts the malware's invocation of network APIs used for D2 connection (Figure 2, ❶). D2 connection is determined by comparing the connection target (e.g., *razy* uses `twitter.com`) against a pre-defined allow list of D2 candidates ❷. This list is based on Tranco [30][1] , allowing us to identify only benign websites that support DDR[2]. After the D2 connection, the malware retrieves the rendezvous point from data hosted by a D2 account (e.g., `pidoras6` for *razy*). Before proceeding with the analysis,

[1]Available at https://tranco-list.eu/list/J49Y.

[2]We assume malicious connection targets if not found on Tranco.

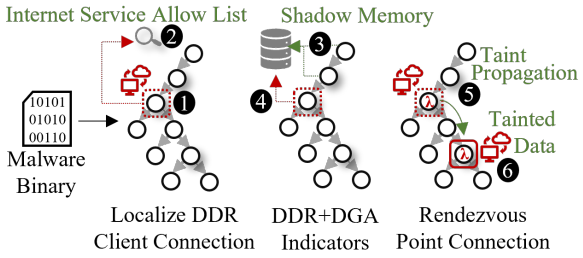

Figure 2: DDR Logic Localization.

R2D2 pinpoints how the malware determined the D2 account. This is important because some DDR malware are hybrids, meaning they use DGAs to generate D2 accounts (DDR+DGA). If we can identify hybrid DDR malware, we can employ similar counteraction techniques against traditional DGA malware.

Finding DDR+DGA Indicators. There are 3 categories of D2 account origins: (1) hard-coded in the malware, (2) retrieved from a dropped file, or (3) dynamically generated, like DGA malware. To categorize, R2D2 populates a shadow memory throughout analysis ❸. When the malware invokes a network API, R2D2 extracts the memory location where the D2 account is and searches the shadow memory for its origin ❹. For (1), R2D2 recursively traces back through the shadow memory to find which instruction last defined the account name's memory location until no more definers are found. In this case, R2D2 ends at a concrete value representing the starting location of the account name in the malware. While (2) is solved in the following section, R2D2 does not explore the following path as no actual account can be retrieved from symbolic data. However, R2D2 continues to identify potential backup accounts.

Category (3) is similar to (1) because a portion of the URL is hard-coded in the malware, but the account is dynamically generated (e.g., `twitter.com` and `1b0xsrs`). DGAs use a seed to initialize account name generation. This seed is often based on a system-available value (e.g., `GetTickCout`), like the popular Conficker DGA malware [42]. Since R2D2 hooks system query APIs and injects symbolic data to bypass defensive evasion, the account name will be symbolic. In contrast, the domain name will be concrete. When R2D2 identifies this symbolic/concrete data, it originates from a DGA.

DGA malware counteraction techniques are effective [21], [23], but they do not enable D2 account origin analysis. R2D2 does not aim to improve DGA counteraction methods but exposes opportunities to counteract hybrid DDR malware, i.e., applying DGA counteraction techniques is not an option until the DGA is known. The location of the last API used to generate the DGA seed pinpoints the location of the DGA in the malware. Whether hybrid or traditional DDR malware, R2D2 next moves to identify the rendezvous point connection.

Table 1: Common Malware Data Manipulation Algorithms.

Decryption & Decoding Algos	References
Exclusive OR (XOR)	[45]–[49]
AES	[50], [51]
DES	[52], [53]
Data Protection API	[53]
RC4	[45]
String to Int, Int to String	[54]
Character Rotation	[47], [48]
Character Subtraction	[48]
Base 16	[37], [48], [55]
Base 32	[56]
Base 64	[36], [37], [47], [48], [57]
Base 85	[58], [59]
String Parsing	[36], [47], [54], [55], [60]

Rendezvous Point Connection. DDR malware retrieves its rendezvous point (e.g., Figure 1) and de-manipulates it to uncover its C&C server address. To confirm, R2D2 injects symbolic (tainted) data into the memory location (e.g., *razy* uses `lpBuffer` of `WinHttpReadData`) that stores the rendezvous point and tags it (Figure 2, ❺). R2D2 uses concolic taint propagation to track the tag, but since R2D2 injects symbolic data for taint analysis, multiple states are spawned for exploration. This is especially computationally expensive when symbolic loops are encountered. However, loops often do not lead to increased code coverage [43], so R2D2 maps explored code regions and references this map when selecting a new state which limits new-state exploration by prioritizing those that lead to unexplored code. 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 [44], we set an upper bound run time to 15 minutes which we find more than sufficient for our study. For R2D2's evaluation, we calculated an average run time of 409 seconds to localize DDR logic, i.e., when the tag appears in an API that is responsible for establishing a network connection (e.g., *razy* uses `pswzServerName` for `WinHttpConnect`) ❻, R2D2 has localized the DDR logic (i.e., *razy* Twitter rendezvous point retrieved using `WinHttpReadData` was used to establish a connection to the C&C server using `WinHttpConnect`).

3.2 De-Manipulation IO Boundary Isolation

Data manipulation includes encryption (e.g., AES) and encoding (e.g., Base16) of rendezvous points. When the malware retrieves the rendezvous point, it *de-manipulates* it, e.g., *razy* (in §2) uses String Parsing and Base64 decoding.

Our preliminary research suggests 14 de-manipulation (9 decoding, 5 decryption) algorithms common in malware (Table 1). R2D2 first isolates boundaries in the symbolic expression (tainted data) where de-manipulation occurs to compare with reference implementations of those in Table 1. If R2D2 can identify the data de-manipulation techniques, authorities can uncover C&C server addresses towards DDR botnet monitoring and counteraction.

Reference Algorithm Implementations. We referenced open-source code repositories and libraries to identify implementations of algorithms in Table 1. R2D2 can be extended by adding algorithm source code. Reference implementations are either (1) integrated into R2D2 so we can submit an input and observe its output, and (2) R2D2 *symbolically* explores the reference implementation resulting in a symbolic expression representing its algorithmic computations. To identify decoder IO boundaries, R2D2 uses (1), and to confirm de-manipulation algorithms from the boundaries, R2D2 uses (2), described in §3.3.

Concerning IO boundaries, symbolic expressions do not have any natural or observable delineations. As the malware executes, injected symbolic data assumes operations corresponding to mathematical computations. Thus, partitioning a symbolic expression is a unique challenge, especially when the malware uses more than 1 de-manipulation algorithm. If algorithms are incorrect or out of order, authorities cannot recover the C&C server address. Thus, R2D2 takes the symbolic expression from DDR logic localization (§3.1) and uses *concrete decoder analysis* (§3.2.1) for decoder isolation and *decryption source-to-sink mapping* (§3.2.2) for decryption isolation. This enables R2D2 to identify the de-manipulation recipe from the symbolic expression (§3.3).

3.2.1 Concrete Decoder Analysis. To isolate decoder boundaries, R2D2 uses Algorithm 1, which relies on concrete values computed during malware execution (concrete values are based on constraints accumulated during execution) to find the start and end (IO) of decoding. As R2D2 analyzes the malware, stepping through instructions in the binary, Algorithm 1 evaluates memory accesses and concretizes its symbolic contents (lines 4-6). Next, the algorithm iterates through decoders from Table 1 and uses the concrete data as input for each (lines 7-8). If the concrete data is the retrieved rendezvous point, then the output is the C&C server address. Each output is stored in a container for algorithm-to-instruction mapping L for later referencing (line 9). Thus far, the concretized data C (line 6), the decoder D (line 7), the decoded result d (line 8), and L (line 9) have been initialized. This process continues throughout R2D2's analysis. Concurrently, the algorithm iterates through all previously stored results in L (lines 10-13) to extract comparative data for boundary isolation. For example, as it iterates through L, if the algorithm finds a previous decoded value (e.g., the C&C server address) that matches with the current concrete value, then the decoder boundary begins at the instruction of the decoded value to the instruction of the current matching concrete value.

We illustrate Algorithm 1 using *razy* in Figure 3. At instruction #4 ($Inst4$), *razy* accesses memory and R2D2 concretizes the contents to *razy's* rendezvous point ① (also see Figure 1). The input is submitted to reference decoder algorithms, and the output ② is stored for later comparison. As execution continues, R2D2 compares concretized results with previously decoded results to identify a match. If a

Algorithm 1 De-Manipulation IO Boundary Isolation

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

1: L : Container to store Algorithm-to-Instruction mapping
2: **function** DeManipulationBoundary($Malware$)
3: **while** $Instruction$ = ExploreMalware($Malware$) **do**
4: $I \leftarrow Instruction$
5: **if** $M \leftarrow MemoryAccessed(I)$ **then**
6: $C \leftarrow ConcretizeMemoryContent(M)$
▷ Iterate 9 Decoder Algorithms
7: **for each** $D \in Decoders$ **do**
8: $d \leftarrow DecodeConcreteData(D, C)$
9: $UpdateDecoderMapping(L, I, C, d)$
▷ Iterate all previously mapped results
10: **for each** $l \leftarrow L$ **do**
11: $I_p, C_p, d_p \leftarrow l$
▷ Previously decoded results equals current concrete value
12: **if** $d_p \equiv C$ **then**
13: $UpdateDeManBoundary(DB, D, I_p, I)$
14: **else if** $C_i \leftarrow ImportedFuncCall(I)$ **then**
▷ Iterate 7 Decryption Algorithms
15: **for each** $D \in Decryptors$ **do**
▷ Locate Start and End decryption boundaries
16: **if** $S, I_s \leftarrow DecryptIO(D, C_i)$ **then**
17: $IBoundary \cup (S, I_s)$
18: **else if** $E, I_e \leftarrow DecryptIO(D, C_i)$ **then**
19: $OBoundary \cup (E, I_e)$
▷ Create Decryption IO Pairs
20: $DecryptIOPairs \leftarrow MatchIO(IBoundary, OBoundary)$
21: **for** $Pr \in DecryptIOPairs$ **do**
22: $D, I_s, I_e \leftarrow Pr$
23: $UpdateDeManBoundary(DB, D, I_s, I_e)$
24: **if** DB **then**
25: $ClosestBoundary(DB)$

match is identified ③, the decoder boundary (e.g., Base64) is from $Inst4$ to $Inst35$ ④. However, R2D2 may identify multiple boundaries for each decoder. Thus, the boundary for Base64 is also $Inst12$ to $Inst35$ ⑤. This occurs if a wrapper function is used that does not modify input or output values. Before completing boundary analysis, R2D2 computes the shortest distance between boundary instructions for boundary isolation (Algorithm 1, line 25).

Using the boundary, R2D2 segments the symbolic expression from §3.1 for verification. An astute reader may consider symbolic expression matching redundant since R2D2 has already concretely delineated the malware's decoders. This is necessary to discriminate 2 algorithms producing the same output, e.g., if *razy's* rendezvous point `aHR0cHM6Ly93MHJtLmluL2pvaW4vam9pbi5waHA=` is decoded with Base64, the result is `https://w0rm.in/join/join.php`. If we use the same input and XOR with `6148523063484d364c79395b393c3a07764243027c401d5808-391b1c0e04575f08065c194f38294d`[3], we arrive at identical outputs. If the DDR botnet orchestrators post additional encoded messages, relying on only the concretely derived decoders could lead authorities to apply the incorrect de-manipulation recipe. Thus, R2D2 compares each segmented boundary with its corresponding reference implementation symbolic expression for verification.

3.2.2 Decryption Source-To-Sink Mapping. R2D2 intercepts invoked API for DDR logic localization (§3.1).

[3]Converted from ASCII to hexadecimal for readability.

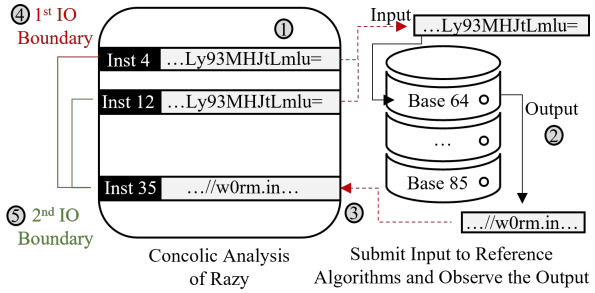

Figure 3: Razy's Decoder IO Boundary Isolation.

These interceptions also prove useful to link decryption APIs-in-sequence for analysis. Using Algorithm 1, while R2D2 explores the malware (line 1), it evaluates call instructions and selects those specific to imported functions (line 14). If the imported function is an API responsible for instantiating decryption (e.g., `CryptAcquireContext`), R2D2 updates the *IBoundary* set representing the boundary source (lines 16-17). If the boundary sink is identified (e.g., `CryptReleaseContext`), R2D2 updates *OBoundary* (lines 18-19). To pair sources and sinks, R2D2 uses $MatchIO$ (line 20). For example, during concolic exploration, if two states are explored, and the first state invokes a source and sink API, those APIs are paired since they were invoked and intercepted by R2D2 from the same state. Lastly, these pairs are used to update our container of all de-manipulation boundaries for further analysis (lines 21-23).

3.3 De-Manipulation Recipe Verification

As discussed in §3.2, though we have de-manipulation algorithm boundaries, symbolic expression matching is still needed for verification. The rendezvous point retrieved from the DDR may contain additional characters used as string markers, or portions of the data may be manipulated differently. If we do not segment the symbolic expression before comparison with a reference implementation, a false negative may occur for de-manipulation recipe identification. Thus, granularity is necessary to verify.

Segmenting Expressions. Algorithm 2 takes DB (decoding boundaries) from Algorithm 1 to segment the symbolic expression (lines 4-6). Each boundary b (lines 4-5) contains the decoder type D and the starting I_s and ending I_e boundary addresses used to segment the expression based on its attributes, i.e., addresses where the symbolic expressions grow. R2D2 parses only those parts of the expression within the boundary (line 6). Now, R2D2 compares the segmented expression with the symbolic reference implementation expression corresponding to D (line 7).

Compare Decoder Expressions. R2D2 injects symbolic data λ to localize the DDR logic which generates a malware symbolic expression M_λ (Figure 4 ❶). To generate the reference implementation symbolic expressions, M_λ is used as input to decoding algorithms ❷, ensuring the resulting

Algorithm 2 Symbolic Expression Matching

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

1: $M_\lambda \leftarrow$ Malware Symbolic Expression
2: $RI \leftarrow$ Symbolic Expressions for Reference Implementations
3: **function** SymbolicExpressionMatching(DB)
4: **for each** $b \in DB$ **do**
▷ Extract the decoder and start & end boundary addresses
5: $D, I_s, I_e \leftarrow b$
6: $s \leftarrow SegmentExpression(M_\lambda, I_s, I_e)$
7: $ri \leftarrow RI.index(D)$ ▷ Assign decoder symbolic expression
8: **if** $dm \leftarrow CompareExpressions(s, ri)$ **then**
9: $DR.add(dm)$

expression (e.g., $B64_\lambda$ ❸) assumes the constraints imposed during the malware execution. This is a prerequisite for symbolic expression comparison, which relies on symbolic solvers. When a decoding algorithm is symbolically explored, concretized values assumed during forking correspond to the previous constraints imposed by M_λ.

Now, having both symbolic expressions, R2D2 can compare them for equivalence ❹. For comparison (line 8), there are 2 conditional constructs: (1) check if the overall expressions match, if not, (2) solve for and compare the concretized outputs. Toward clarity, we use a symbolic expression with starting λ values `(Read byte_1, v0_xor_0)`. This expression is used as input to 2 versions of an XOR by `0x23` algorithm. As the expression is decoded, it grows with additional operations corresponding to algorithmic computations. For example, 1 byte of M_λ and XOR_λ is partially transformed into:

M_λ = `(Or (ZExt (Read byte_1, v0_xor_0)) 0x23)`

XOR_λ = `(Xor 0x23 (ZExt (Read byte_1, v0_xor_0)))`

Comparing M_λ and XOR_λ considers node placement, edges, and expression size. This static check is less computationally expensive. If it fails, R2D2 invokes the symbolic solver to compare concretized outputs of both expressions. Concrete values are based on the constraints from M_λ that are imposed on XOR_λ during its execution. This ensures expressions are evaluated based on the same constraints resulting in the same concrete output if they are functionally identical. Since each comparison is per explored path, the results are a ratio of matches versus non-matches, i.e., when concretized outputs for each path are equal, it is a match, else a non-match. If the ratio is high enough (further explained in §4), their equivalence is confirmed.

Compare Decryption Expressions. At this point, R2D2 carefully verifies decryption algorithms. R2D2 partitions boundaries based on source and sink markers. Specifically, the boundaries are based on constraints imposed on sinks (*OBoundary*) as they relate to their source (*IBoundary*). Recall, R2D2 uses taint analysis for DDR logic localization (§3.1). Although not a DDR malware, *loadmoney's* cryptographic function (left of Figure 5) illustrates segmenting symbolic expressions based on an isolated boundary. From steps 1-2, *loadmoney* acquires the handle to the cryptographic service protocol before retrieving the hash object. From 3-6, *loadmoney* computes the message hash for later comparison. The dotted lines in Figure 5 match the

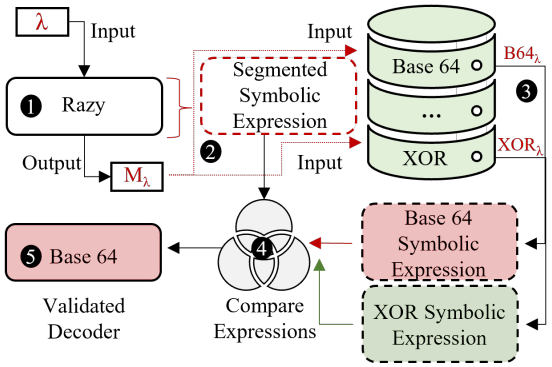

Figure 4: R2D2 Symbolic Expression Matching for Razy.

input [in] constraints of a parameter with the output [out] constraints of at least one API predecessor. When a sink decryption API is invoked (e.g., `CryptReleaseContext`), R2D2 backward analyzes the constraints to recover the API sequence confirming the decryption algorithm. This is known as *decryption constraint chaining* since the APIs recovered contain constraints that can be *linked* to preceding APIs. The second example illustrates a potential DDR malware employing decryption APIs. After data is read from the D2 (e.g., *razy* uses `twitter.com`) (7), decryption begins with cryptographic context initialization (8) before key derivation (9). Then, the data read (`lpBuffer`, 10) is decrypted. R2D2's backward analysis would confirm the constraints of the decryption algorithm revealing the APIs-in-sequence, which are used to segment the malware's symbolic expression, i.e., extract only the portion of expression relating to decryption.

To generate reference implementation symbolic expressions for comparison with the malware expression, R2D2 uses decryption models, i.e., source code implementations for decryption algorithms from Table 1 based on the sequence of APIs. Like the previously described symbolic exploration of decoding functions, R2D2 explores the decryption algorithm implementations to generate a symbolic expression. R2D2 uses decryption constraint chaining to identify constraints of interests that relate only to the sequence of APIs. Then, R2D2 uses *compareExpression* (Algorithm 2, line 8) to match the segmented expression with constraints corresponding to decryption reference implementations. If they are the same, their constraints will match because R2D2 is only interested in the algorithmic computations as they pertain to decryption API invocation. If there is a confirmed match, R2D2 reports the de-manipulation recipe containing an ordered set of decryptors used in malware.

4 VALIDATING OUR TECHNIQUES

R2D2 is implemented in C++/Python leveraging S2E [41] for concolic analysis. We use custom code (≈8k LoC) to localize the DDR logic and identify the de-manipulation recipe.

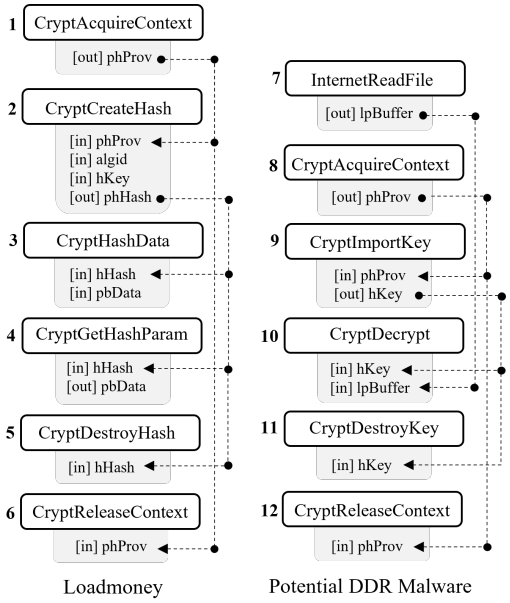

Figure 5: Constraint Chaining for Crypto Identification.

4.1 DDR Logic Localization

To stress-test howR2D2 localized the DDR logic, we randomly chose 5 known DDR malware families and 5 variants of each [4] totalling 25 ground truth DDR malware. Since we also seek to identify hybrid DDR+DGA malware, we used a known DGA malware, *clocker* (or *cryptolocker*, last row), to augment our ground truth dataset.

Table 2 presents R2D2's ground truth performance. Columns 1-2 list malware families and the number of variants. The next 8 columns are the ground truth D2s and D2 account origins, including accuracy metrics — true positive (TP), false positive (FP), and false negative (FN), respectively. The final column list the average exploration time for each malware. Overall, R2D2 correctly localized the DDR logic for 23/25 DDR malware with a 95% Acc. (accuracy). For D2 account origin, R2D2 achieved a 96% (28/30) accuracy. Furthermore, of the 30 malware, 10 of them had dead endpoints (*comnie* and *clocker*). However, unlike pure concrete execution, which cannot analyze malware with non-available endpoints, R2D2 enabled malware exploration irrespective of endpoint liveness.

Upon closer inspection, we found 2 FNs in R2D2's DDR logic localization of *msil* (Row 5). A manual investigation confirmed this as a DDR malware, but R2D2 could not locate the rendezvous point connection. This resulted from an unresolved symbolic constraint accumulated through loop exploration. Although R2D2 prioritizes code exploration from unexplored regions (§3.1), this multi-nested loop exhausted R2D2's 15-minute run time. However, this rarely occurred. We also note 2 FNs during *clocker* analysis which occurred

[4]There is a 93.75% chance that the median value of a DDR malware is between the smallest and largest values in any random sample of five.

Table 2: Validating DDR Logic Localization.

Family	#	Dead Drop	TP	FP	FN	D2A Origin	TP	FP	FN	Time(s)
razy	5	twitter.com	5	0	0	HC	5	0	0	148
doina	5	drive.google.com	5	0	0	HC	5	0	0	230
kryptik	5	pastebin.com	5	0	0	HC	5	0	0	484
comnie	5	github.com[1]	5	0	0	HC	5	0	0	417
msil	5	pastebin.com	3	0	2	HC	5	0	0	803
clocker	5	nwbyrkswt.net[1,2]	-	-	-	DGA	3	0	2	332
Total	30	Acc. (95%)	23	0	2	Acc. (96%)	28	0	2	409

D2A = D2 Account, HC = hard-coded

1: Dead D2 identities (e.g., `pidoras6` for *razy* from §2)
2: This is one of many DGA domains that R2D2 identified

because 2 variants contained a hard-coded target that the malware attempted to connect to. If it failed, then the DGA was used. R2D2 rightly classified the hard-coded target and thus did not consider the DGA component in 2/5 malware.

Given the low number of FNs (4) and the high accuracy of 95% and 96% for DDR logic localization and D2 account origin, respectively, R2D2 is ready for large-scale deployment. Moreover, averaging a run time of 409 seconds, R2D2 can quickly report details of DDR integration.

4.2 De-Manipulation Recipe Identification

Given 2 different implementations of the same algorithm, will R2D2 still be able to identify that algorithm? Put simply, will R2D2's reference implementations (from Table 1) work regardless of a malware's implementation? Answering this requires (1) comparing source code similarity to ensure all algorithm implementations are different and (2) comparing symbolic expression equivalence of all pairs of algorithms to show that irrespective of implementation, de-manipulation algorithms of the same class (e.g., all XOR decryptors) match with high confidence.

Source Code Similarity. We chose up to 3 implementations of algorithms in Table 1 from open-source software repositories or libraries. Suppose we can show that symbolic expression matching finds equivalence in algorithms of the same class with differing implementations. If so, R2D2's approach will work irrespective of a malware's implementation. We compared all combinations of the 35 algorithm implementations using Moss [61], a software similarity framework widely used in academia to detect code plagiarism, which reported 0% match in all comparisons of different implementations (1,190 comparisons). All algorithms compared with themselves (35 comparisons) were a 100% match. We present detailed results in Appendix B.

Symbolic Expression Matching. We compare all similar algorithms (i.e., algorithms that rely on APIs are compared with one another, see Table 3a, and others are likewise compared, see Table 3b and Table 3c). A pairwise comparison of expressions is not limited to 1 evaluation. When expressions are compared, their concrete output comparison is per path explored (§3.3). Since we specify a symbolic input size of 8 bytes, the algorithm's expressions

Table 3: De-Manipulation Algorithm Similarity.

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

(a) AES - RC4.

		XOR			Chr Sub			S ⇆ I		Chr Rot			Str Prs	
		v1	v2	v3	v1	v2	v3	→	←	v1	v2	v3	v1	v2
XOR	v1	100	100	100	11	0	0	11	0	0	0	0	0	0
	v2	100	100	100	0	0	0	11	0	0	0	0	0	0
	v3	100	100	100	0	0	0	11	0	0	0	0	0	0
Chr Sub	v1	0	0	0	100	92	94	0	0	0	0	0	0	0
	v2	0	0	0	86	100	84	11	0	0	0	0	0	0
	v3	0	0	0	92	91	100	11	0	0	0	0	0	0
Str, Int	→	11	11	11	1	0	0	100	4	0	0	0	0	0
	←	0	0	0	0	0	0	22	100	0	0	0	0	0
Chr Rot	v1	6	4	6	2	6	0	0	0	100	100	100	0	0
	v2	0	0	0	0	0	0	1	0	100	100	100	0	0
	v3	0	0	0	0	0	0	1	0	100	100	100	0	0
Str Prs	v1	0	0	0	1	0	0	11	0	13	3	3	100	99
	v2	0	0	0	2	0	0	11	0	19	2	3	99	100

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

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

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

could concretize to 256^8, or 4.2 billion, possible inputs or paths. Ideally, we would evaluate the entire input space, but that is time prohibitive. So, we compare each pair for 2 hours. We find this to be sufficient because the ratio of non-matches versus matches plateau and stabilizes for more than 30 minutes within 2 hours, convincing us of the overall percentage match. In the worse case, when Base16 is compared with Base85 (Table 3c, Row 1, Column 4), it plateaus at 85 minutes. We present a view of the plateau for each set of comparisons in Appendix C.

We also found implementations of the same algorithm that produce different outputs given the same input. This is expected in algorithms that do not rely on built-in libraries like decoders and the XOR decryptor, which are not rigorously tested to ensure completeness or error handling. Still, these differences are negligible compared to those with other algorithm classes. To illustrate, Table 3c cells represent the expression matching percentage. The majority ($474/554$[5]) of comparisons of algorithms from different classes result in 0% match. However, notice Base64 v1 compared with the other versions (Row 7). We expect them to have a high match rate, even amid mismatching corner cases. Our investigation revealed that the 44% and 77% rates for v2 and v3 are due to error checking in v1. When R2D2 forks at this error check, it will take 2 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 exploring the first algorithm. So, where v1 had 1 failure and 1 success, v2 has less strict error checking and 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 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 closer match.

Table 3c also shows an interesting trend within the Base decoding classes (Base 16, etc.). We expect that 12/144 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 expect 108 with a 0% match (when 1 Base class is compared with another). Base algorithms depend on a table of data for character translation. As a result, all Base algorithms match to a certain extent based on these tables, but not enough to deduce a confident equivalence. There are other overlapping algorithms from differing classes, but they are too low to be considered a match. Lastly, we notice the ideal algorithm equivalence case from Table 3a. Since 5/6 of the decryption algorithms in this study rely on built-in libraries, their implementations are very similar. Thus, R2D2 considers the algorithm identifier when available to differentiate decryptors when their symbolic expressions are identical. This ensures a more accurate match and rightly distinguishes decryptors in Table 3a.

From each class, we select implementations that match >90% within their class and <25% across the others (e.g., v1). These are built into R2D2 as "reference implementations", but new algorithms can always be added to R2D2. These results show that R2D2 can identify de-manipulating algorithms with different implementations in malware.

4.3 De-Manipulation Recipe Identification

To test symbolic expression matching, we augment our ground truth dataset with 2 non-DDR malware families (*spora* and *muldrop*) that include cryptographic routines. Table 4 lists our results, including a combination of the IO boundary isolation and verification of the de-manipulation algorithm. In all cases, the IO boundary and verified algorithm matched,

[5] 71/625 algorithms are compared with themselves.

Table 4: Validating De-Manipulation Recipe Identification.

Family	#	Algorithm	TP	FP	FN
razy	5	String Parsing, Base64	5	0	0
doina	5	String Parsing	0	5	0
kryptik	5	String Parsing, Base64	5	0	0
comnie	5	Base64, Char Rotate	5	0	0
spora[1]	5	AES	5	0	0
muldrop[1]	5	RC4	5	0	1
Total	30	Acc. (89%)	25	5	1

1: Our investigation did not uncover DDR malware decrypting rendezvous points.

i.e., there were no instances where multiple algorithms were identified in the same boundary. Columns 1 and 2 lists the malware family and the number of samples. Column 3 lists the de-manipulation recipe. The final columns present the accuracy metrics TP, FP, and FN. Overall, R2D2 correctly identified the IO boundary and verified the de-manipulation recipe for 25 malware with 89% accuracy.

Our investigation revealed 5 FPs. We expect data manipulation of rendezvous points, but *doina* requests a Google Drive text file containing a plaintext IP address. However, R2D2 identified String Parsing. This occurs because the malware checks for the IP address format using string search routines, similar to what is needed for string parsing. As a reminder, *razy* used string parsing to remove the preceding `)))))` characters before it could use Base64 to decode the remainder of the string (§2). In that case, string parsing was integral to decoding, unlike in *doina*. R2D2 may be prone to FPs when dealing with plaintext data, so R2D2's 89% accuracy is less reflective of the true impact of de-manipulation recipe identification since plaintext rendezvous points need no de-manipulation and authorities will immediately recover the C&C address. Next, similar to the FNs from Table 2, R2D2 encountered unresolved symbolic constraints during the analysis of *muldrop*, meaning that it could not leverage constraint chaining for description source-to-sink mapping (§3.3). However, given the low number of FPs and FNs, R2D2 maintains a high accuracy of de-manipulation recipe identification.

5 INSIGHTS INTO DDR MALWARE

In this section, we discuss our findings from deploying R2D2 on 100K malware. This dataset was built by downloading Windows executables with more than one anti-virus (AV) engine detection and an upload date between 2017 and 2022 to VirusTotal [62]. The dataset parameters are derived from Zhu et al. [63], who provided a comprehensive study of VirusTotal labeling dynamics. Since it is difficult to distinguish traffic generated by DDR malware from benign network traffic, we also ensured that each sample connects to at least 1 endpoint listed in Tranco [30] to validate R2D2's ability to correctly distinguish D2 connections that appear benign.

Now, we present the DDR malware found (§5.1) and the de-manipulation algorithms identified (§5.2). To provide context for the impact of these findings, we geolocate rendezvous points and dead drops (§5.3) and also provide a case study to illustrate how R2D2 enables DDR malware counteraction (§5.4). Finally, we discuss our efforts to counteract DDR malware in collaboration with service providers (§5.5).

Table 5: DDR Malware and Dead Drops.

Dead Drops Domains	#Fam	#Mal	D2A Origin	
			HC	DGA
blockchain.info	59	1,888	1,888	0
blockcypher.com	41	1,437	1437	0
pastebin.com	30	6,053	6,053	0
bitaps.com	16	722	722	0
docs.google.com	10	616	616	0
coinmarketcap.com	10	50	50	0
googleusercontent.com	9	151	151	0
twitter.com	8	34	22	12
blockchain.com	7	5	5	0
dropbox.com	6	204	204	0
blockr.io	5	200	200	0
github.com	3	7	7	0
blockstream.info	3	3	3	0
wordpress.com	2	4	4	0
drive.google.com	1	3	3	0
Total	154	10,170[1]	10,158	12

D2A: D2 Account, HC: Hard-Coded, DGA: DGA+DDR Hybid

1: This is the sum of unique DDR malware but not the sum of the above column. Most malware using blockchain D2s use more than 1 D2 as a backup, so they appear in multiple rows.

5.1 DDR Malware Findings

Table 5, Column 1 lists the 15 D2s identified. Columns 2-3 list the number of malware families and variants. Column *D2A Origin* lists the number of malware containing hard-coded (traditional DDR) versus DGA (hybrid DDR) accounts. R2D2 identified 10,170 DDR malware across 154 families that maliciously use 275 accounts in our dataset.

Blockchain.info D2 represents the most malware families (Row 1) at 1,888 (≈38% of families and ≈19% of malware) in our dataset. In fact, there are 7 different blockchain D2s (Rows 1-2, 4, 6, 9, 11, and 13), totaling 3,098 DDR malware. Interestingly, 1,054 of those use more than 1 D2 for backup, the only category of DDR we found using backups. However, each malware uses 1 wallet ID to retrieve the rendezvous point. Appendix D lists of the 75 identified wallet IDs.

Pastebin is the third most prevalent service in our dataset, accounting for ≈20% of malware families across 6,053 malware (≈59%). Pastebin has long been used for malicious purposes [64], but it has generally been used to host stolen content or dropper malware. This work is among the first to expose its pervasiveness as a dead drop. R2D2 also identified Twitter, Google, Github, and Dropbox. However, they account for fewer occurrences than expected, totaling 1,019 or ≈10% of all DDR malware. Several works have studied the abuse of these services (§6) though none

Table 6: De-Manipulation Algorithms Mapped to DDRs.

	#De-Manipulation Algos/Malware							
Dead Drops	**Str Prs**	**Base64**	**Base16**	**Int to Str**	**XOR**	**Char Rot**	**Plain Text**	**Total**
Blockchain	3,098	0	3,098	3,098	0	0	0	9,294
Pastebin	6,053	4,813	0	0	872	0	368	11,738
Google Docs	616	479	0	0	0	123	137	1,095
GContent[1]	151	151	0	0	0	67	0	302
Google Drive	3	0	0	0	0	0	3	3
Twitter	34	34	0	0	0	0	0	68
Dropbox	204	0	0	0	204	204	0	408
GitHub	7	0	7	0	7	7	0	21
WordPress	4	4	0	0	0	0	0	8
Total	10,170	5,481	3,105	3,098	872	211	508	22,937[2]

1: Google User Content.
2: An average of 2.25 de-manipulation algorithms per malware.

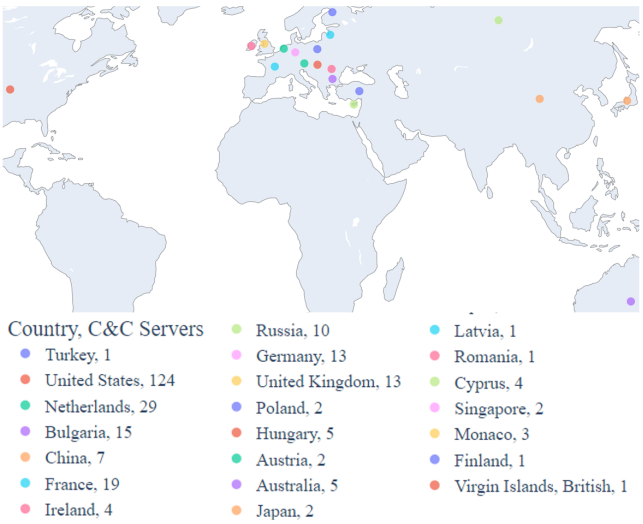

Figure 6: Worldview of Geolocalized Rendezvous Points.

considered D2s, so malware authors are seemingly using less popular internet services to reduce suspicion.

Next, 12 DDR malware, 0.12% of our dataset, use a DGA to generate Twitter accounts (Column *D2A Origin*). This is the *fugrafa* malware and is similar to *miniduke* [37] in its DDR capability. Although this malware complexity can pose challenges for authorities, the plethora of works that counteract DGA malware may motivate malware authors to opt for the traditional and not the hybrid DDR approach.

Recently, Netskope reported that over 66% of malware downloads come from internet services [31]. However, what was unknown is how many use these services as D2s, which R2D2 finds to be 10,170, or ≈10% of our dataset.

5.2 De-Manipulation Algorithms Identified

Table 6 maps de-manipulation algorithms with their corresponding D2. Column 1 lists the dead drops, and Columns 2-7 lists the de-manipulation algorithms. Of 9 decoders from Table 1, R2D2 found 5 being used by malware. Of the 5 cryptographic algorithms we built into R2D2, only XOR is used by malware. This is expected as encrypting rendezvous points requires malware authors to trade agility for more complex malware. This is the opposite of the trend that we and others [40], [65] have observed.

From the Total row, the most common de-manipulation algorithms are String Parsing and Base64 decoding, occurring in 10,170 and 5,481 DDR malware, respectively. Base64 provides obfuscation and ensures data goes unmodified during transport, adding to its popularity. We also notice that D2s hosting visible rendezvous points (e.g., Twitter's publicly viewable messages) generally use data manipulation. For example, Pastebin DDR malware prefer String Parsing (in 6,053 samples), Base64 (in 4,813 samples), and XOR (in 872 samples), obscuring the public posts.

D2s like Google Drive host content that can only be viewed if a user has the exact URL to access it, which is often a long string of randomized characters. Thus, malware authors are more likely to store plaintext rendezvous points on these D2s, and R2D2 identified ≈5% of DDR malware doing so, Column 8 (*Plain Text*). Notably, 508 samples in Column 8 are also counted as String Parsing in Column 2. From §4.3, R2D2 identifies String Parsing for plaintext rendezvous points.

Finally, de-manipulation becomes trivial if the correct algorithm is identified. Yet, malware authors use them because it is difficult to identify the manipulation algorithm, especially when multiple are used. Column *Total* reveals that each malware uses, on average, 2.25 de-manipulation algorithms (22,937/10,170). R2D2 can detect this layered approach so authorities can de-manipulate these more complex rendezvous points toward counteraction.

5.3 Geolocalized Rendezvous Points

To illustrate the benefits of rendezvous point retrieval and de-manipulation provided by R2D2, Figure 6 shows the geolocation of C&C servers found *behind* the resolved and decoded rendezvous points at the time of this study. The figure includes dots of various colors representing a country where a C&C server is based and the number of C&C servers per country. R2D2 geolocalized the C&C servers via the WhoisXML API [66], which includes 4+ billion domains and subdomains and records for over 99% of IPs in use.

From Figure 6, we notice that the United States accounts for the most C&C servers, with 124 unique C&C servers found. Again, this is not the victim D2 service but the subsequent C&C server that the decoded rendezvous point points to. The US may seem like an outlier compared to the quantity of C&C servers found worldwide, but existing research agrees that attackers evade attribution and sometimes detection by distributing their C&C servers near their intended targets [67].

Digging deeper into these results, we observe the trends between the abused D2s and their C&C server location. As shown in Table 7, Column 3 (*US*) reveals that all services

Table 7: C&C Server Count Across Dead Drops by Country.

Dead Drops	Turkey	US	Netherlands	Bulgaria	China	France	Ireland	Russia	Germany	UK	Poland	Hungary	Austria	Australia	Japan	Latvia	Romania	Cyprus	Singapore	Monaco	Finland	British VI
Blockchain	1	62	22	15	2	15	2	5	11	7	2	5	2	5	2	1	1	4	2	3	1	1
Pastebin	-	24	6	-	3	-	2	5	2	-	-	-	-	-	-	-	-	-	-	-	-	-
Google Docs	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
GContent	-	10	-	-	1	2	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-
Google Drive	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Twitter	-	7	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
Dropbox	-	7	1	-	1	2	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-
GitHub	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
WordPress	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 8: R2D2's Razy Analysis Results.

DDR Logic	Traditional DDR malware (not hybrid)
Localization	`http://twitter.com/pidoras6`
De-Manipulation Recipe	String Parsing, Base64 Decoding Bytes [0:4] Bytes [5:n] `)))))` `aHR0cHM6Ly93MHJtLmluL2pvaW4vam9pbi5waHA=`
Counteraction Options	(1-2) `https://w0rm.in/join/join.php` (3) Prepend `)))))` to `Ym90bmV0LnNpbmtob2xlLmNvbQ==`[1]

1: `botnet.sinkhole.com`

that enable D2s lead to C&C servers in the US. Next, the Netherlands and France (Columns 4 and 7) show the second most C&C servers using dead drops at 29 and 19, respectively. Although Pastebin is used by most DDR malware samples (6,053 from Table 5), they resolve to a smaller set of C&C servers (42). The geological distribution shows that blockchain apps are used in all countries identified, with 171 C&C servers located. This is expected as bitcoin transactions attract criminals who erroneously rely on its anonymity [68].

5.4 Razy Reloaded: A Case Study

The *razy* case study illustrates the capabilities authorities have when armed with R2D2. Now, authorities have several options to counteract DDR malware: (1) de-manipulate the rendezvous point to locate and seize the C&C server and then request that the service provider disables the account; (2) track C&C server migration, i.e., after C&C server seizure, the botnet orchestrators can migrate to a new C&C server by updating the rendezvous point. Authorities can track migration if service providers leave the account active for botnet monitoring. Lastly, authorities, in collaboration with service providers, can (3) replace the rendezvous point to sinkhole or take over the botnet.

Table 8 shows R2D2's analysis of *razy* and provides data to pursue either of the 3 counteraction options. Firstly, DDR logic localization begins with a connection to Twitter and the `pidoras6` account (Row 1). *Razy* then retrieves the rendezvous point from the `pidoras6` post and begins de-manipulation. As shown in Row 2, R2D2 identifies String Parsing and Base64 decoding for the de-manipulation recipe. To correctly apply the recipe, R2D2 notes that String Parsing corresponds the first five bytes ($[0:4]$) of the rendezvous point and Base64 corresponds to the remainder ($[5:n]$). For String parsing, R2D2 identified 5 leading close parentheses symbols `)))))` hard-coded into the malware to be parsed from the rendezvous point. Since the D2 endpoint is still live, authorities can retrieve the rendezvous point and de-manipulate it to `https://w0rm.in/join/join.php`. Authorities are now armed (Row 3) to seize the C&C server.

For counteraction open 1, using the C&C server address, authorities can locate the hosting provider and request support in disabling connectivity. This approach is widely used [8], [23]–[26], [69]. At the same time, authorities can request that service providers disable the D2 account to prevent botnet resurgence. For option 2, instead of requesting the D2 account be disabled, authorities monitor the D2 account after C&C server seizure to observe any new botnet activity. Botnet monitoring has previously been shown effective at information gathering toward disruption or takedodwn [40]BDSdo we have more for this cite list?. Similarly, authorities can monitor DDR botnets to understand the geographic scale of infection or indicators that may lead to botnet orchestrator attribution.

For option 3, authorities can leverage the recipe in reverse, i.e., use encoding to generate a sinkhole address in the correct format to masquerade as the rendezvous point. For example, if authorities want to direct bots to `botnet.sinkhole.com`, they would use Base64 *encoding* resulting in `Ym90bmV0LnNpbmtob2xlLmNvbQ==`. Finally, authorities would prepend the parsed string to the Base64 encoded string: `)))))Ym90bmV0LnNpbmtob2xlLmNvbQ==`. In conjunction with C&C server seizure, this approach not only dismantles the existing botnet but redirects new bots to a dead-end preventing further botnet proliferation.

R2D2 enables authorities to identify DDR malware, extract evidence of malicious internet platform use for cooperation from providers, and pursue counteraction.

5.5 Towards Remediation

We lack the authority to pursue domain seizures or sinkholing, so we sought cooperation from the affected service providers.

Counteraction Confirmed. R2D2 found 275 D2 accounts, and we reported them to the service providers, who confirmed our findings and took action against 9,155 DDRs (90% of our total findings). Specifically, Pastebin disabled (Appendix E) the offending accounts, resulting in at least 6,053 infected victim systems unable to communicate with the D2 account for migration once the C&C server is added to a block list (60% of all DDR malware that R2D2 identified). Moreover, any new cots cannot access the C&C server. Similarly, WordPress also responded to our findings and removed the offending accounts from their platform (Appendix E). Although WordPress DDR malware only accounted for 0.04% of DDR malware discovered, it illustrates the range of options malware authors have.

For the 3,098 DDR malware (30%) that use bitcoin wallets to retrieve rendezvous points, they must connect to 1 of many

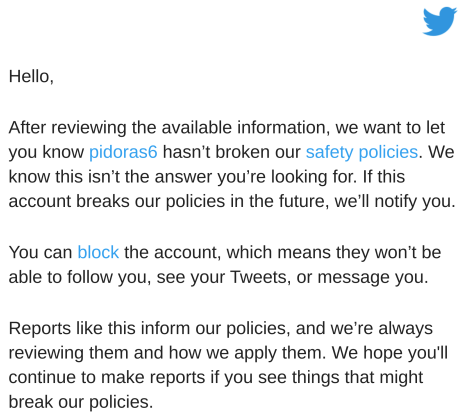

Figure 7: Response From Twitter.

blockchain apps (e.g., `blockchain.info`) to search and read the blockchain. Disk space requirements make downloading the entire blockchain to the victim system impractical. Thus, although blockchains are immutable, these apps control access to view blockchain transactions. A practical solution for 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 (Appendix D) that R2D2 discovered to BitcoinAbuse [70] to publicly document that they have been used in malware.

Counteraction In Progress. Service providers are faced with a difficult task: weighing freedom of expression while ensuring users adhere to the terms and conditions [71], [72] and also ensuring they do not unnecessarily block content on their platforms. Although providers are culpable for hosting malicious content [73], we noted the difficulty of convincing providers that harmless-looking content hosted on their platforms drives malware. For instance, while Pastebin quickly removed offending content, WordPress required more correspondence to convince them of malicious platform use. We also encountered cases where providers were uncooperative in remediating DDR behavior on their platform. Since service providers maintain sole control, we are limited to their actions against DDR botnets. Notably, 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 Figure 7. Of the remainder, the accounts for 774 malware were already taken down at the time of our study. We are awaiting a response from the service providers hosting the remaining 241 DDRs.

6 RELATED WORK

Botnet Counteraction. Existing methods block malicious network traffic to counteract botnets [6]–[11], [39], [40]. However, traffic generated by DDR malware appears benign [1], [2], and many malware often access benign websites to test internet connectivity [6]. So, even if prior works are extended to target malware that rely on benign network traffic, they cannot distinguish D2 traffic from other benign traffic. Other works propose solutions to recover encoded C&C URLs from a binary [12]–[14]. However, none of these can recover dynamically resolved C&C server URLs or IPs, preventing their application to DDR malware. Subsequent research have counteracted DGA malware [23]–[26], but these approaches focus on in-binary C&C server address generation algorithms. Unlike DGA malware, DDR malware C&C server addresses are solely hosted outside of the binary, preventing the transference of DGA counteraction techniques in this emerging domain.

Malicious Internet Platform Use. Existing research investigates social network abuse [74]–[76]. Ohters have analyzed network traffic to identify abused cloud repositories [77] or cloud app abuse to enable botnets [27]–[29], [78]. These works did not pursue cross-domain analysis (malware and abused internet platform) to fully understand DDR malware or counteraction measures. Next, researchers also investigated malware using bitcoin wallets or cloud apps to coordinate C&C server access [3]–[5], [33], [37], [55], [57], [79]. Still, they focus on one malware family or opt for network or bitcoin transaction analysis. R2D2 identifies this capability from malware analysis, which is necessary to gain full insights into the attack. Moreover, R2D2 also provides authorities with actionable means to counteract DDR botnets.

Concolic Analysis. Concolic analysis is used to find software bugs [41], [80], [81], generate test cases [82]–[84], support dynamic analysis [85]–[87], and taint analysis [88]–[91]. However, pure concolic analysis does not allow analysis irrespective of malware defensive evasion or endpoint liveness. Conversely, R2D2 leverages modeled APIs to ensure continued execution and DDR logic localization. Next, other works have used the input to output domain mapping for binary similarity and bug hunting via symbolic analysis [92]–[95] or dynamic analysis [96]–[99]. However, R2D2 moves beyond comparing concrete input and output values. After identifying areas of interest in malware, R2D2 extracts the symbolic expression within a boundary defined by input and output mappings to match symbolic expressions for de-manipulation recipe identification.

7 CONCLUSION

This paper used R2D2 to comprehensively study the under-explored DDR technique and analyzed 100K malware spanning back 5 years. R2D2 revealing 10,170 DDR malware from 154 families. R2D2 also identified rendezvous point de-manipulation recipes enabling authorities to uncover C&C server addresses, with String Parsing and Base64 being the most common. We reported our findings to service providers, who took action against the 9,155 DDRs (90% of DDR malware discovered). Of the remainder, service providers previously took down the accounts for 774 malware, and we are awaiting a response concerning 241 DDRs.

REFERENCES

[1] *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].

[2] *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].

[3] 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.

[4] S. Pletinckx, C. Trap, and C. Doerr, "Malware Coordination Using the Blockchain: An Analysis of the Cerber Ransomware," in *2018 IEEE Conference on Communications and Network security (CNS)*, 2018.

[5] Z. Huang, J. Huang, and T. Zang, "Leopard: Understanding the Threat of Blockchain Domain Name Based Malware," in *International Conference on Passive and Active Network Measurement*, Springer, 2020, pp. 55–70.

[6] G. Jacob, R. Hund, C. Kruegel, and T. Holz, "Jackstraws: Picking Command and Control Connections from Bot Traffic," in *Proceedings of the 20th USENIX Security Symposium (Security)*, San Francisco, CA, Aug. 2011.

[7] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou II, and D. Dagon, "Detecting Malware Domains at the Upper DNS Hierarchy," in *Proceedings of the 20th USENIX Security Symposium (Security)*, San Francisco, CA, Aug. 2011.

[8] 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].

[9] I. Arghire, *TrickBot Botnet Survives Takedown Attempt*, https://www.securityweek.com/trickbot-botnet-survives-takedown-attempt, [Accessed: 2020-12-10].

[10] J. Menn, *Court orders seizure of ransomware botnet controls as U.S. election nears*, https://www.reuters.com/article/us-uselection-cyber-botnet/court-orders-seizure-of-ransomware-botnet-controls-as-u-s-election-nears-idUSKBN26X1G2, [Accessed: 2022-07-18].

[11] C. Page, *Rsocks, A popular proxy service, was just seized by the DOJ*, https://techcrunch.com/2022/06/17/rsocks-proxy-seized-justice-department/, [Accessed: 2022-07-18].

[12] C. Zuo and Z. Lin, "Smartgen: Exposing Server Urls of Mobile Apps with Selective Symbolic Execution," in *Proceedings of the 26th International World Wide Web Conference (WWW)*, Perth, Australia, 2017.

[13] L. Glanz, P. Müller, L. Baumgärtner, *et al.*, "Hidden in Plain Sight: Obfuscated Strings Threatening Your Privacy," in *Proceedings of the 15th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Taipei, Taiwan, Oct. 2020, pp. 694–707.

[14] M. Y. Wong and D. Lie, "Tackling Runtime-Based Obfuscation in Android with TIRO," in *Proceedings of the 27th USENIX Security Symposium (Security)*, Baltimore, MD, Aug. 2018.

[15] *Fast Flux Networks Working and Detection*, https://resources.infosecinstitute.com/topic/fast-flux-networks-working-detection-part-1, [Accessed: 2022-03-12].

[16] *Dynamic Res.: Fast Flux DNS*, https://attack.mitre.org/techniques/T1568/001/, [Accessed: 2022-03-12].

[17] *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].

[18] *Dynamic Res.: DNS Calculation*, https://attack.mitre.org/techniques/T1568/003/, [Accessed: 2022-03-12].

[19] *Whois Numbered Panda*, https://www.crowdstrike.com/blog/whois-numbered-panda/, [Accessed: 2022-03-12].

[20] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, "A Comprehensive Measurement Study of Domain Generating Malware," in *Proceedings of the 25th USENIX Security Symposium (Security)*, Austin, TX, Aug. 2016, pp. 263–278.

[21] M. Antonakakis, R. Perdisci, Y. Nadji, *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.

[22] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis.," in *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2011, pp. 1–17.

[23] V. Le Pochat, S. Maroofi, T. Van Goethem, *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.

[24] 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.

[25] R. Wainwright and F. J. Cilluffo, *Responding to Cybercrime at Scale: Operation Avalanche — A Case Study*, http://www.jstor.org/stable/resrep20752.

[26] B. Stone-Gross, M. Cova, L. Cavallaro, *et al.*, "Your Botnet is my Botnet: Analysis of a Botnet Takeover," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, Chicago, Illinois, Nov. 2009, pp. 635–647.

[27] M. Torkashvan and H. Haghighi, "CB2C: A Cloud-Based Botnet Command and Control," *Indian Journal of Science and Technology*, 2015.

[28] 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.

[29] 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.

[30] 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.

[31] *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].

[32] *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].

[33] *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.

[34] *Attack Matrix for Enterprise*, https://attack.mitre.org/, [Accessed: 2021-11-06].

[35] *Web Service*, https://attack.mitre.org/techniques/T1102/, [Accessed: 2021-11-06].

[36] *Analyzing malware by API calls*, https://blog.malwarebytes.com/threat-analysis/2017/10/analyzing-malware-by-api-calls/, [Accessed: 2022-03-06].

[37] *Operation Ghost. The Dukes aren't back — they never left*, https://www.welivesecurity.com/wp-content/uploads/2019/10/ESET_Operation_Ghost_Dukes.pdf, [Accessed: 2022-02-26.

[38] B. Cheng, J. Ming, J. Fu, *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 25th ACM Conference on Computer and Communications Security (CCS)*, Toronto, ON, Canada, Oct. 2018.

[39] O. Alrawi, M. Ike, M. Pruett, *et al.*, "Forecasting Malware Capabilities From Cyber Attack Memory Images," in *Proceedings of the 30th USENIX Security Symposium (Security)*, Virtual Conference, Aug. 2021.

[40] J. Fuller, R. P. Kasturi, A. Sikder, *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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] *Lazarus Targets Latin American Financial Companies*, https://www.trendmicro.com/en_us/research/18/k/lazarus-continues-heists-mounts-attacks-on-financial-organizations-in-latin-america.html, [Accessed: 2022-11-15].

[46] *Double Dragon - APT41, a Dual Espionage and Cyber Crime Operation*, https://content.fireeye.com/apt-41/rpt-apt41, [Accessed: 2022-03-06].

[47] *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].

[48] *Understanding the Patchwork Cyberespionage Group*, https://documents.trendmicro.com/assets/tech-brief-untangling-the-patchwork-cyberespionage-group.pdf, [Accessed: 2022-03-06].

[49] *Bronze Butler Targets Japanese Enterprises*, https://www.secureworks.com/research/bronze-butler-targets-japanese-businesses, [Accessed: 2022-03-06].

[50] *Explained: Spora ransomware*, https://www.malwarebytes.com/blog/news/2017/03/spora-ransomware, [Accessed: 2022-11-15].

[51] *Operation Endtrade: Tick's Multi-Stage Backdoors for Attacking Industries and Stealing Classified Data*, https://documents.trendmicro.com/assets/pdf/Operation-ENDTRADE-TICK-s-Multi-Stage-Backdoors-for-Attacking-Industries-and-Stealing-Classified-Data.pdf, [Accessed: 2022-11-15].

[52] *Analysis of TeleBots' cunning backdoor*, https://www.welivesecurity.com/2017/07/04/analysis-of-telebots-cunning-backdoor/, [Accessed: 2022-11-15].

[53] *Invisimole: The Hidden Part of the Story Unearthing Invisimole's Espionage Toolset and Strategic Cooperations*, https://www.welivesecurity.com/wp-content/uploads/2020/06/ESET_InvisiMole.pdf, [Accessed: 2022-11-15].

[54] *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].

[55] *Casbaneiro: Dangerous Cooking with a Secret Ingredient*, https://www.welivesecurity.com/2019/10/03/casbaneiro-trojan-dangerous-cooking/, [Accessed: 2022-03-06].

[56] *Multigrain - Point of Sale Attackers Make an Unhealthy Adition to the Pantry*, https://www.fireeye.com/blog/threat-research/2016/04/multigrain_pointo.html, [Accessed: 2021-11-06].

[57] *The Dropping Elephant - Aggressive Cyber-Espionage in the Asian Region*, https://securelist.com/the-dropping-elephant-actor/75328/, [Accessed: 2022-03-06].

[58] *Gustuff Banking Botnet Targets Australia*, https://blog.talosintelligence.com/2019/04/gustuff-targets-australia.html, [Accessed: 2022-03-06].

[59] *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, [Accessed: 2021-11-06].

[60] *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].

[61] *A System for Detecting Software Similarity*, https://theory.stanford.edu/~aiken/moss/, [Accessed: 2022-02-26.

[62] *VirusTotal*, https://www.virustotal.com/, [Accessed: 2022-1-5].

[63] S. Zhu, J. Shi, L. Yang, *et al.*, "Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines," in *Proceedings of the 29th USENIX Security Symposium (Security)*, Virtual Conference, Aug. 2020.

[64] *The Malicious Use of Pastebin*, https://www.fortinet.com/blog/threat-research/malicious-use-of-pastebin, [Accessed: 2022-03-12].

[65] *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].

[66] *WhoisXML API*, https://whois.whoisxmlapi.com/, [Accessed: 2022-11-10].

[67] *Malware C&C Servers Found in 184 Countries*, https://threatpost.com/malware-cc-servers-found-in-184-countries/99870/, [Accessed: 2022-12-11].

[68] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies," in *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, May 2015, pp. 104–121.

[69] C. Rossow, D. Andriesse, T. Werner, *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.

[70] *Bitcoin Abuse Database*, https://www.bitcoinabuse.com/, [Accessed: 2022-2-06].

[71] J. A. Pater, M. K. Kim, E. D. Mynatt, and C. Fiesler, "Characterizations of Online Harassment: Comparing Policies Across Social Media Platforms," in *in Proceedings of the 19th international conference on supporting group work*, 2016, pp. 369–374.

[72] S. Zannettou, J. Blackburn, E. De Cristofaro, M. Sirivianos, and G. Stringhini, "Understanding Web Archiving Services and their (Mis) Use on Social Media," in *in Proceedings of the 12th International AAAI Conference on Web and Social Media*, 2018.

[73] K. Thomas, D. Akhawe, M. Bailey, *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.

[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] S. Alrwais, K. Yuan, E. Alowaisheq, Z. Li, and X. Wang, "Understanding the dark side of domain parking," in *Proceedings of the 23rd USENIX Security Symposium (Security)*, San Diego, CA, Aug. 2014.

[79] G. Gomez, P. Moreno-Sanchez, and J. Caballero, "Watch your back: Identifying cybercrime financial relationships in bitcoin through back-and-forth exploration," in *Proceedings of the 29th ACM Conference on Computer and Communications Security (CCS)*, Los Angeles, CA, Nov. 2022, pp. 1291–1305.

[80] 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.
[81] 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.
[82] J. C. King, "Symbolic Execution and Program Testing," 7, vol. 19, ACM, 1976, pp. 385–394.
[83] R. S. Boyer, B. Elspas, and K. N. Levitt, "SELECT — A Formal System for Testing and Debugging Programs by Symbolic Execution," ACM, 1975.
[84] L. A. Clarke, "A System to Generate Test Data and Symbolically Execute Programs," *IEEE Transactions on Software Engineering*, no. 3, pp. 215–222, 1976.
[85] 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.
[86] K. Kim, I. L. Kim, C. H. Kim, *et al.*, "J-force: Forced Execution on Javascript," in *Proceedings of the 26th International World Wide Web Conference (WWW)*, Perth, Australia, 2017, pp. 897–906.
[87] 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.
[88] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (But Might Have Been Afraid to Ask)," in *Proceedings of the 31th IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 2010, pp. 317–331.
[89] M. G. Kang, S. McCamant, P. Poosankam, and D. Song, "Dta++: Dynamic Taint Analysis With Targeted Control-Flow Propagation," in *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2011.
[90] J. Newsome and D. X. Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," in *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS)*, vol. 5, San Diego, CA, Feb. 2005, pp. 3–4.
[91] J. Clause, W. Li, and A. Orso, "Dytan: A Generic Dynamic Taint Analysis Framework," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, London, UK, Jul. 2007.
[92] J. Ming, D. Xu, Y. Jiang, and D. Wu, "BinSim: Trace-based Semantic Binary Diffing via System Call Sliced Segment Equivalence Checking," pp. 253–270.
[93] D. Xu, J. Ming, and D. Wu, "Cryptographic Function Detection in Obfuscated Binaries via Bit-Precise Symbolic Loop Mapping," in *Proceedings of the 38th IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, May 2017.
[94] Y. David, N. Partush, and E. Yahav, "Statistical similarity of binaries," *Acm Sigplan Notices*, vol. 51, no. 6, pp. 266–280, 2016.
[95] L. Luo, J. Ming, D. Wu, P. Liu, and S. Zhu, "Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection," in *in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 389–400.
[96] M. Chandramohan, Y. Xue, Z. Xu, Y. Liu, C. Y. Cho, and H. B. K. Tan, "Bingo: Cross-architecture cross-os binary search," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 678–689.
[97] J. Pewny, B. Garmany, R. Gawlik, C. Rossow, and T. Holz, "Cross-architecture bug search in binary executables," in *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, May 2015, pp. 709–724.
[98] S. Wang and D. Wu, "In-memory fuzzing for binary code similarity analysis," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2017, pp. 319–330.
[99] M. Egele, M. Woo, P. Chapman, and D. Brumley, "Blanket execution: Dynamic similarity testing for program binaries and components," in *Proceedings of the 23rd USENIX Security Symposium (Security)*, San Diego, CA, Aug. 2014, pp. 303–317.
[100] *Malpedia: Free and Open Malware Reverse Engineering Resource offered by Fraunhofer FKIE*, https : / / malpedia . caad . fkie . fraunhofer . de, [Accessed: 2021-11-06].
[101] 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.
[102] 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,

A ANTI-ANALYSIS APIS

Malware use anti-analysis or defensive evasion APIs to evade detection. R2D2 hooks these APIs to enable continued malware exploration. The complete list of defensive evasion APIs considered in designing R2D2 is listed in Table 9. These APIs are based on our manual malware analysis, reports from industry experts [34], [100], and prior research investigating victim system emulation and malware evasion techniques [44], [101], [102].

Table 9: Defensive Evasion APIs.

Anti-Analysis and Defensive Evasion APIs	
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

B DE-MANIPULATION ALGORITHM SOURCE CODE SIMILARITY

We use Moss [61] to ensure each decoding algorithm considered in R2D2 is markedly different. As shown in Table 10, no differing algorithms overlap, as formalized in Row 6, where $|R|$ is the length of the set containing all de-manipulation algorithm implementations and m and n

Table 10: Baseline Comparison for De-Manipulation Algorithm C/C++ Source Code via Moss [61].

	Base16			Base32			Base64			Base85			Char Sub.			Str ⇆ Int		Rotate			Str Prs		XOR			AES		DES		DPAPI		RC4	
	v1	v2	v3	v1	v2	v3	v1	v2	v3	v1	v2	v3	v1	v2	v3	S→I	S←	v1	v2	v3	v1	v2	v1	v2	v3	v1	v2	v1	v2	v1	v2	v1	v2
v1	100	0	0	100	0	0	100	0	0	100	0	0	100	0	0	100	0	100	0	0	100	0	100	0	0	100	0	100	0	100	0	100	0
v2	0	100	0	0	100	0	0	100	0	0	100	0	0	100	0	0	100	0	100	0	0	100	0	100	0	0	100	0	100	0	100	0	100
v3	0	0	100	0	0	100	0	0	100	0	0	100	0	0	100	-	-	0	0	100	-	-	0	0	100	-	-	-	-	-	-	-	-

$\forall m, n \in 0, |R| \wedge m \neq n : MossR_n, R_m = \varnothing$

are indices of different algorithms in the set. The only matches occurred when 1 algorithm was compared with itself, which is expected. This verifies that each de-manipulation algorithm selected for R2D2 is a different implementation.

C DE-MANINPULATION COMPARISON GRAPHS

Table 11 displays graphs for each set of de-manipulation algorithm comparisons. For example, in Column 1, Row 1, the graph consists of comparisons for all 3 versions of Base 16 totaling 9 comparisons (i.e., Base16v1 vs Base16v2, etc.). The graph displays a 2-hour run time (X-axis) where each line corresponds to a de-manipulation similarity (Y-axis) comparison. From §4.2, we picked 2 hours because we observed the match rate for each comparison plateau for at least 30 minutes, giving us confidence that the value at the plateau is the most accurate view of equivalence. The black vertical line marks the latest plateauing comparison and shows that all comparisons plateau for more than 30 minutes. In this case, the earliest plateau occurred at 20 minutes and the latest plateau at 85 minutes. We select the best matching algorithm from each class for R2D2 to use for malware de-manipulation algorithm identification.

D WALLET IDS

R2D2 found 75 bitcoin wallet IDs in 3,098 malware, as shown in Table 13 and Table 12. These wallet IDs were reported as malicious toward remediation. However, since R2D2 reveals how the blockchain transactions are decoded to C&C server IP addresses, authorities can submit transactions to the offending accounts to sinkhole the botnet [3].

E SERVICE PROVIDER COLLABORATION

Figure 8 and Figure 9 shows Pastebin's and WordPress' response. Within 48 hours, the offending accounts were removed.

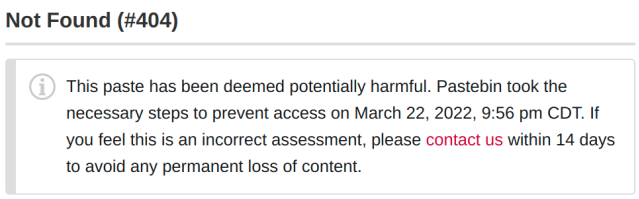

Figure 8: A Pastebin Account Removed.

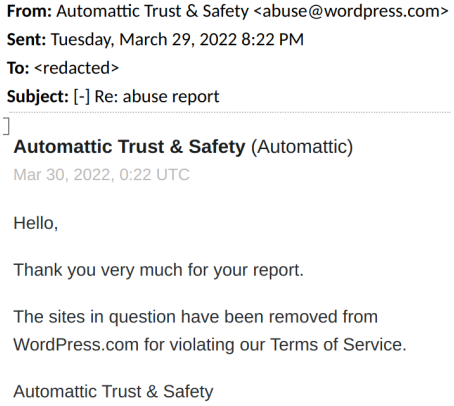

Figure 9: Response From WordPress.

Table 11: Baseline Comparison for Decoding Algorithm Similarity via Symbolic Expressions.

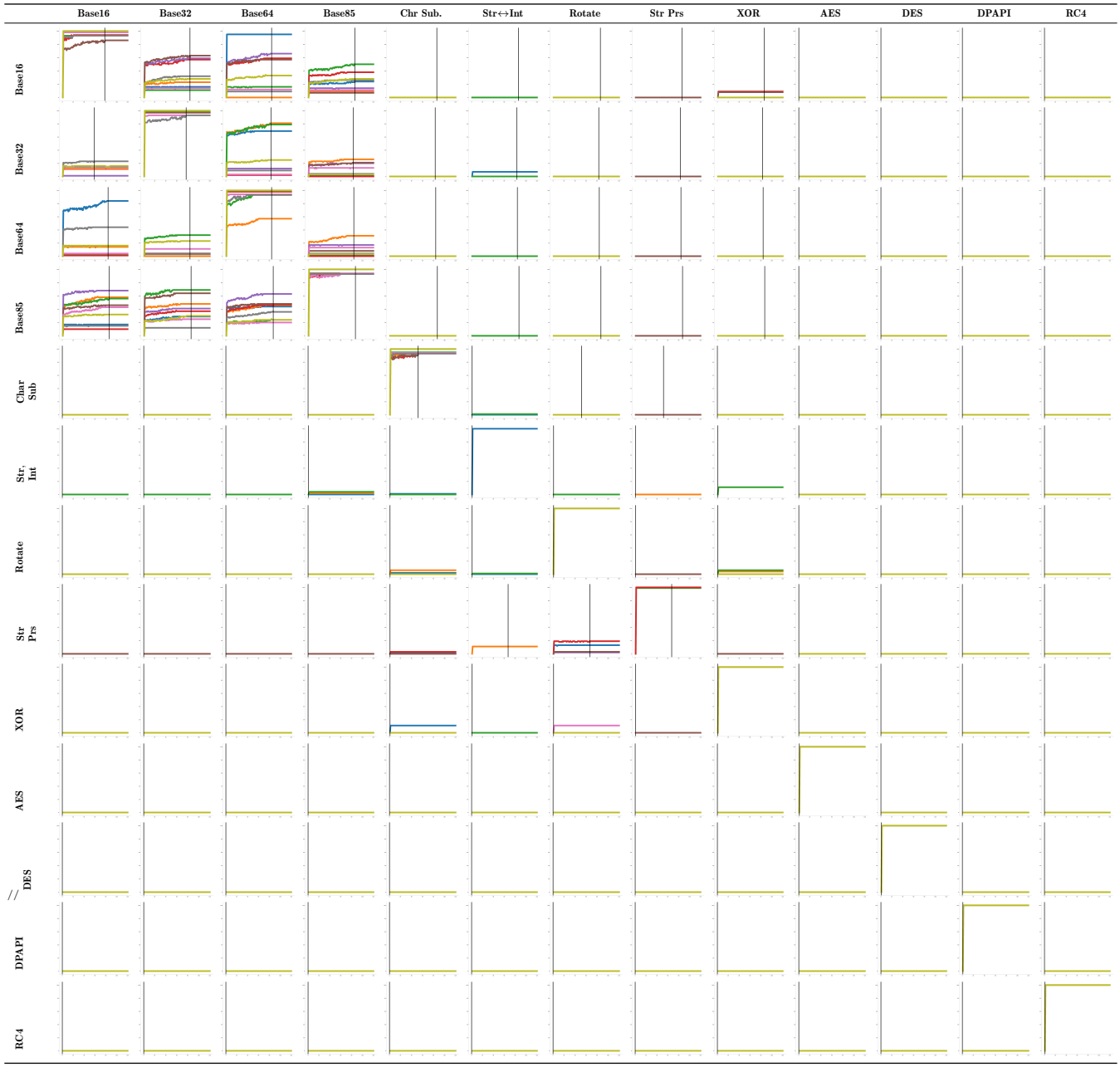

Table 12: Bitcoin Wallet IDs.

Blockchain App	Wallet ID
coinmarketcap.com	1BkeGqpo8M5KNVYXW3obmQt1R58zXAqLBQ 1N94rYBBCZSnLoK56omRkAPRFrpr5t8C1y
blockstream.info	1qre9cdrqdagy0p2sww2dvp7td86kws09v 1qj2h87z0v8u7ddp823apvjzpu5asssfpy
blockchain.info	1Lud76Q98VRHCUiyK7XUs7AgFofrqXeP78 15GqSWnxEFZezUCcGjhBMknA1PB7aYNXC1 18sHYU49vUFk6TN6G2Pj6DSCUzkbLvwJtc 1DsyxmgvkBTnLBnCXWyymNaDNhgmzib4mp 1BkeGqpo8M5KNVYXW3obmQt1R58zXAqLBQ 13LHbsf1CWgat1ZLYYoMsjeeybvCD7ZUxh 3ab5ab9511cf52565314425424d0b0b978 1qtkmks24vyuemjm6w3j3qagyn2pu3d93y 1NL67bQ8dPbfxLKcXBpuE3n8H5AsExBvwt 38D2P6apsGhghkGK4mSAMB9yr5enXW6iUy 32Lsw4r5YGLS5qhZsgp1b2kk1xTbf7T4Wf 16nA62oxxsDgc2R2NoW6WtFrZkB3XLvVpb 323c2a4e57b5ba21687fe7ce5918ceaf4a 1Dhf71bPe3wQ4At9YSaEVXGyhwzFiKNdBo 1q97764a4dnuzfd5dxxyhqggyn7de9z978 1qsce320qf73s9v593p0jxfs46q7nh0zus 1NxsR82Efaqbnt3c9QQUoYJpejwFtDrnNe 1Eyx9PKb1bs9X7n4UK7JHnzSxedyUvHimE 1FK6Y4BcHV5jQ6nPJTL83EyrhLzWGTvkfv 1PwAEK6Zp371Vegwp38XzP4nULikzUrCRa 1K4GnSGtoH7qx4SvpoJ3v3Nv2yZg6v5sDY 1EYnNjRKWqFBRLe53Ui1HVwxDwK1gqsKGp 3Gf7NRXDKtAzeTYG9fwHLg9snpSi2AZpyj 3QKjFKdqtJi34UnFwdA8VsaV2NkRgmaUnf 3NXdpbSZqe3sniegAgzEpo7YisDqVQiL6y 1KVwMFZw4QUuoqdeWtehDeB7qLhx3ncGVV 1KQheJU4ZwXvMdoLevVhDVVHU1zwWUMcDp 1qwqr922tvn69gsdhn7fcayspwt202fzer 33u5t5A8qvQFdwVMANDFS91LGMw68fiaMc 1GAEMH9wiX8LqJm7v8oKLdgM5Zr3msE6E3
blockcypher.com	1HTDy9SkfhwaNCXFA8wFCvN53f3iGpm8kb 1016d7ceff188e9fe32e68e9761bd811f3 17gd1msp5FnMcEMF1MitTNSsYs7w7AQyCt 1BkeGqpo8M5KNVYXW3obmQt1R58zXAqLBQ 3916a96ba7bfc95ed103aff4286360e820 1ML94w1SCudkiFHaEwYqTmKGTkywxVBuZg 1a4778fd2ba2b8ae98c573defa3b0e86d8 1GcnsLs7C31uuroNmUHwwbB5xQeNvm63Ee 193896af781481f195a4c55cbf053b7e95 1CMRScsrxPe2N4HwPpNKcHUhfCJXUm2Cx6 3bcec9103e14bd8969f2d1f2e14bd72399 3e3c52d8aeed29d2e5f2835061f01fa758 18ecb27aff6d1c6a889f810c50eb72e565 1d8c213480d883fc2c4a001ecfb106f241 1CpTCVckjajNKDd7PsApV3cAkunVd4Mcmt 198009d287c818d2a9aa72f7f828c19c84 19hi8BJ7HxKK45aLVdMbzE6oTSW5mGYC82 1N9ALZUgqYzFQGDXvMY5j1c7PGMMGYqUde 133be6e6ccca5bb6b3e3aaa34cf14a374a 34d153ae12ebfe18cea39ddc07d514865b 14bbtRSruiXHtvofYgB24Wdpma1Bx6RSof 19ZN4JM9ZH2nLc3PZh85n3t1WVzjBKD39D 14a0b3c26dc368d1b69862eb28fd8648fd 1ALuqPer2DSD9YyU9nrZz6NR1dDwCQLnE7 1BYZgQnu3M86ra95Jywj5xiL2fE7Nbn64q 1Fbhv84haM4TiwcR71WCVZg87EWbFxFUZC 1Q5qfq1tC7ptd5bGWimbCkJT1hp8v9eNfu 33fde6e00a62995ddd4977b5cf7b8bc55c 1VocauiabutZLvzBau7V6QgCC7WQnmU1n2 1BjVeaZBMA9QEweeRfK6nftzDPbr7jMaDk 161fPjdCt5H9uYawPpPT4poc8RBcLFaE3R 1CeLgFDu917tgtunhJZ6BA2YdR559Boy9Y
bitaps.com	17gd1msp5FnMcEMF1MitTNSsYs7w7AQyCt 1CMRScsrxPe2N4HwPpNKcHUhfCJXUm2Cx6 1HTDy9SkfhwaNCXFA8wFCvN53f3iGpm8kb

Table 13: Bitcoin Wallet IDs.

Blockchain App	Wallet ID
blockr.io	1cptcvckjajnkdd7psapv3cakunvd4mcmt 17gd1msp5FnMcEMF1MitTNSsYs7w7AQyCt 1HTDy9SkfhwaNCXFA8wFCvN53f3iGpm8kb 1a4778fd2ba2b8ae98c573defa3b0e86d8 1GcnsLs7C31uuroNmUHwwbB5xQeNvm63Ee 1CpTCVckjajNKDd7PsApV3cAkunVd4Mcmt 1d6037414ac2bbf101eadae4d4c4d57e98 1CMRScsrxPe2N4HwPpNKcHUhfCJXUm2Cx6 17bf8ba6d1bb9e5f03b0946d467fca7887 1b354ee81d0ea177be5355b8a430db1b22 3a0ed3db93838620fee7aed8c87f3ebacb 11486a040f0cf7511a53f7610958f2a109 35ef6c71fafa9e30ee56d312dd626999ac 1e77054da43c04f19b628a7ea5bfc6d1ca 1e52ba07c17cc49995c915209b23b23ad6